Spatial Normalization and analysis of Brain MRI Studies – A Deep Neural Network Based Approach

Tiago Rafael Andrade Jesus

UMinho | 2019

Universidade do Minho
Escola de Engenharia

Tiago Rafael Andrade Jesus

Spatial Normalization and analysis of Brain MRI Studies – A Deep Neural Network Based Approach

outubro de 2019

Universidade do Minho
Escola de Engenharia

Tiago Rafael Andrade Jesus

Spatial Normalization and analysis of Brain MRI Studies – A Deep Neural Network Based Approach

Dissertação de Mestrado
Mestrado Integrado em Engenharia Biomédica
Ramo de Informática Médica

Trabalho efetuado sob a orientação de:
Victor Alves
Ricardo Magalhães

outubro de 2019

DECLARATION

Name: Tiago Rafael Andrade Jesus

Dissertation Title: Spatial Normalization and analysis of Brain MRI Studies – A Deep Neural Network Based Approach

Dissertation Title in Portuguese: Normalização espacial e análise de imagens cerebrais de Ressonância Magnética – Uma abordagem com Deep Learning

Mentors: Victor Alves, Ricardo Magalhães

Conclusion Year: 2019

Master Designation: Mestrado Integrado em Engenharia Biomédica

Master Branch: Informática Médica

Universidade do Minho, ____/____/______

Signature: ____

ACKNOWLEDGEMENTS

This dissertation involved a lot of work and would not be possible without the proper guidance.
I would like to start by thanking my cousin, João Sousa, for showing me how incredible the area of Biomedical Engineering is and for always helping me when I needed. And a huge thanks to my cousin Marlene Reis for finding time to help me in her busy schedule. Thank you so much.
I would also like to thank my mentors, Victor Alves and Ricardo Magalhães, for making this thesis a reality.
I want to thank Professor Victor Alves for his exceptional guidance and knowledge. He was always available and ready to meet me. Thank you for all the time spent chatting about my dissertation or solving any other problem that appeared but above all for believing in me. I would also like to thank him for the readiness to accept my request to also guide me through my future PhD thesis, I'm so grateful.
I want to thank Ricardo Magalhães, and his team in ICVS, for the guidance and knowledge in the MRI area. Thank you for guiding me, especially in the early stage of the thesis, and for showing me the incredible area of neuroimaging, I'm truly fascinated by it.
I would like to thank my parents, grandparents, sister and girlfriend for always being there for me and motivating me to finish this dissertation. I wouldn't have gone this far without their support.
I want to dedicate this thesis to everyone who believed in me, especially the ones mentioned above, and I hope I make them all proud.

“ Always remember, however, that there's usually a simpler and better way to do something than the first way that pops into your head. ”

Donald Ervin Knuth (1986). “The METAFONTbook”, Addison-Wesley

STATEMENT OF INTEGRITY

ABSTRACT

Throughout the years, Deep Learning has proven to be an excellent technology to solve problems that would otherwise be too complex. Furthermore, it has shown great success in the area of medical imaging, especially when applied to segmentation of brain tissues. As such, this dissertation explores a possible new approach, using Deep Artificial Neural Networks to perform spatial normalization on brain MRI studies as well as classify using Brain MRI studies regarding their state of brain atrophy.

Spatial normalization of Magnetic Resonance images by tools like the FSL, or SPM turned out to be inefficient for researches as they need too many resources to achieve good results. These resources include, for example, wasted human and computer time when executing the commands to normalize and waiting for the process to finish, this can take up to several hours just for one study. Therefore, a new approach was needed, a faster and easier way to normalize the MRI studies. To do so, Deep Artificial Neural Networks were used by creating a python program to deal with said studies in much less time. This program should free the researchers' time for other more relevant tasks and help reach conclusions faster in their studies when trying to find patterns between the analysed brains. Several architectures were tried, having better results with U-Net based architecture as well as GAN architecture.

At the end, the model couldn't learn correctly all the brain features to be changed in any of the approaches but showed great potential. Even though the final model did achieve the correct shape it could not yet achieve the final normalization.

With some more time invested in perfecting the models, these could, in the future, learn to correctly perform the final normalization and allow the researchers to perform it in less than 10 seconds per exam instead of hours.

Regarding the Brain Atrophy models, the models showed some potential too as the predictions were partially correct. With more data, and less unbalanced, the model could probably learn correctly and output the expected results for all classes.

Keywords: Brain, Deep Learning, MRI, Neuroimaging, Spatial Normalization.

Resumo

Ao longo dos anos, abordagens Deep Learning têm provado ser uma excelente tecnologia para resolver problemas que seriam complexos demais. Além disso, demonstrou grande sucesso na área da imagem médica, principalmente quando aplicada em segmentação de imagens. Como tal, esta dissertação explora uma possível nova abordagem, usando as Redes Neurais Artificiais Profundas para realizar a normalização espacial em estudos de RM do cérebro, bem como classificá-las usando estudos cerebrais de RM em relação ao seu estado de atrofia cerebral.

A normalização espacial dos estudos de ressonância magnética através de ferramentas como a biblioteca FSL acabou sendo pouco eficiente para uso na investigação, pois estas ferramentas precisam de muitos recursos para obter bons resultados. Esses recursos incluem, por exemplo, desperdício de tempo humano e de computador ao executar os comandos para normalizar e aguardar a conclusão do processo; o que pode demorar várias horas, apenas para um estudo. Portanto, uma nova abordagem é necessária, uma maneira mais rápida e fácil de normalizar os estudos de RM. Para isso, foram utilizadas Redes Neurais Artificiais Profundas, criando um programa em python para lidar com os estudos em muito menos tempo. Esse programa deve liberar o tempo dos investigadores para outras tarefas mais exigentes e ajudar a chegar a conclusões mais rapidamente nos seus estudos, ao tentar encontrar padrões entre os cérebros analisados. Várias arquiteturas para o modelo foram testadas, obtendo melhores resultados com a arquitetura baseada em U-Net e com a arquitetura GAN.

No final, o modelo não conseguiu aprender corretamente todos os detalhes do cérebro a serem alterados em nenhuma das abordagens, mas mostrou grande potencial. Apesar de o modelo final ter atingido a forma correta, ainda não conseguiu a normalização final.

Com mais tempo investido no aperfeiçoamento dos modelos, estes poderiam, no futuro, aprender a executar corretamente a normalização final e permitir que os pesquisadores realizassem este processo em menos de 10 segundos por exame, em vez de horas.

Em relação aos modelos de atrofia cerebral, estes também mostraram algum potencial, pois as previsões estavam parcialmente corretas. Com mais dados e menos desequilíbrio nos mesmos, o modelo provavelmente poderia aprender corretamente e gerar os resultados esperados para todas as classes.

Palavras-chave: Cérebro, Neuroimagem, Normalização Espacial, Redes Neurais Artificiais, Ressonância Magnética.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Abreviations and Acronyms

A

Adam	Adaptive Moment Estimation
ADC	Apparent Diffusion Coefficient
AE	Autoencoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
AUC	Area Under the Curve

B

BET	Brain Extraction Tool
BOLD	Blood Oxygen Level Dependent

C

CNN	Convolution Neural Networks
CPU	Central Processing Unit
CSF	Cerebrospinal Fluid
CSV	Comma-Separated Values
CT	Computer Tomography
CUDA	Compute Unified Device Architecture

D

DICOM	Digital Imaging and Communications in Medicine
DL	Deep Learning
DSC	Dice Similarity Coefficient
DSR	Design Science Research
DTI	Diffusion Tensor Imaging

E

EEG	Electroencephalography

F

FCN	Fully convolutional network

FLAIR	Fluid-Attenuated Inversion Recovery
FLIRT	FMRIB's Linear Image Registration Tool
FN	False Negative
FNIRT	FMRIB's Non-linear Image Registration Tool
FP	False Positive
FSL	FMRIB Software Library

G

GAN	Generative Adversarial Network
GB	Gigabyte
GM	Grey Matter
GPU	Graphics Processing Unit

H

HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol

I

ID	Identifier
IDE	Integrated Development Environment

L

LR	Learning Rate
LTS	Long Term Support
LXC	Linux Containers

M

MEG	Magnetoencephalography
ML	Machine Learning
MLP	Multi-Layer Perceptron
MRI	Magnetic Resonance Imaging
MTA	Medial Temporal lobe Atrophy score

N

NIfTI	Neuroimaging Informatics Technology Initiative
NM	Nuclear Medicine

NN	Neural Networks

O

OS	Operating System

P

PET	Positron-Emission Tomography

R

ReLU	Rectified Linear Unit
RMSProp	Root Mean Square Propagation

S

SFTP	Secure File Transfer Protocol
SGD	Stochastic Gradient Descent

T

TB	Terabyte
TN	True Negative
TP	True Positive

U

UI	User Interface

V

VAE	Variational Autoencoder
VM	Virtual Machine

W

WM	White Matter
WWW	World Wide Web

X

XML	eXtensible Markup Language
XNAT	eXtensible Neuroimaging Archive Toolkit
XSD	XML Schema Document

GLOSSARY

Activation Function	Used in Neural Networks, this is the function in the neuron also called the threshold function. It takes in the weighted sum of all of the inputs from the previous layer and then generates and passes an output value (typically nonlinear) to the next layer.
Artificial Intelligence (AI)	Computer programs designed to solve difficult problems which humans (and animals) routinely solve. It allows machines to make decisions and perform tasks that simulate human intelligence and behavior.
Artificial Neural Networks (ANN)	A model that, taking inspiration from the brain, is composed of layers (at least one of which is hidden) consisting of simple connected units or neurons followed by nonlinearities.
Batch	In Artificial Neural Networks, it is a set of examples or samples used in one iteration.
Class	In a Machine Learning context, a class refers to the output category of the data. A label in a dataset points to one of the classes.
Convolutional Neural Network (CNN)	A CNN are a Deep Artificial Neural Network that is currently the state-of-the-art in image processing. Its major advantage is the little pre-processing steps required when compared to other image classification algorithms.
Dataset	A collection of examples. Each example contains one or more features and a label (if using supervised training).
Deep Learning (DL)	A subset of AI and Machine learning which allows computational models composed of multiple processing layers to learn representations of data with multiple levels of abstraction.
DICOM	DICOM (Digital Imaging and Communications in Medicine) is a standard for handling, storing, printing, and transmitting information in medical imaging. It includes a file format definition and a network communications protocol.
Dropout	A form of regularization useful in training Artificial Neural Networks. Dropout regularization works by removing a random selection of a fixed number of the units in a network layer for a single gradient step. The more units dropped out, the stronger the regularization.
Design Science Research (DSR)	The DSR methodology is a rigorous scientific research methodology that is well known to be effective in the area of computer science and medical informatics.

Epoch	A full training pass over the entire data set such that each example has been seen once.
Feature	In a Machine Learning context, a feature is the specification of an attribute and its value. It expresses a piece of measurable information about something.
FSL	A library of analysis tools for FMRI, MRI and DTI brain imaging data. This library contains the tools used in the spatial normalization process.
Machine Learning (ML)	The use of data-driven algorithms that perform better as they have more data to work with. The fundamental goal of ML is to generalize beyond the examples in the training set.
Magnetic Resonance Imaging (MRI)	Non-invasive imaging modality used for disease detection and treatment monitoring. MRI scanners are particularly well suited to image the soft tissues of the body e.g., brain, spinal cord and nerves, muscles and ligaments.
NiBabel	Python package that provides easy read and write access to some of the common neuroimaging file formats. Among the accepted formats are the NIfTI format, the format of the MRI images of the dataset used.
NIfTI	A simple, minimalistic format which has been widely adopted in neuroimaging research, allowing scientists to mix and match image processing and analysis tools developed by different teams.
Overfitting	Creating a model that matches the training data so closely that the model fails to make correct predictions on new data.
Test set	The subset of the dataset that is used to test the model after the model has gone through initial vetting by the validation set.
Train set	The subset of the dataset used to train a model.
Validation set	A subset of the dataset — disjunct from the training set—that is used to adjust hyperparameters.
Voxel	Unit of graphic information that defines a point in three-dimensional space.
XNAT	An open source imaging informatics software platform designed to facilitate common management and handling of neuroimaging and associated data.

1 Introduction

1.1 CONTEXT AND MOTIVATION

Medical informatics was once considered as a ‘nice-to-have’ rather than ‘need-to-have’ as it is becoming nowadays. During the last decade, the field has grown steadily and is now one of the foundations of medicine and health care [1].

Medical imaging, as its name implies, is a field that deals with the process of creating visual representation of the interior of the human body for medical purposes. Several types of medical imaging modalities exist, such as MRI (Magnetic Resonance Imaging), whose images can be functional, molecular or structural depending on the objective of the study, Computed Tomography (CT) which is based on X-rays and Positron Emission Tomography (PET). Several of these modalities can also be combined, using medical image fusion to take advantage of the best parts of each modality [2].

Medical Image Analysis is a part of the field of Computer Vision. It emerged in the early 1990s when researchers began to apply methods that solved problems in other areas to medical images [3].

One of the possible applications of the modalities mentioned above is the acquisition of brain images (neuroimaging), which can be very helpful for different types of studies. Neuroimaging data can help understand how the brain functions, how its different areas respond to different stimuli, or help diagnose multiple psychiatric disorders and diseases, such as tumors.

For example, the paper “Scanning patients with tasks they can perform” [4] studies psychologically impaired patients and presents an overview of types of imaging experiments that can be performed on them. Another paper, “Mapping function in the human brain with Magnetoencephalography, anatomical Magnetic Resonance Imaging, and functional Magnetic Resonance Imaging” [5], makes it possible to take advantage of the different pros each modality has, specifically, EEG, MRI and MEG (Magnetoencephalography). By combining the different strengths of these modalities, the individual limitations of the mentioned neuroimaging techniques can be overcome and thus much faster conclusions can be drawn.

Regarding MRI scans, specifically the brain studies, sometimes a spatial normalization of the brain is required. This normalization essentially makes it possible for the researchers to take conclusions of the brain areas among several brains as it computes them all into the same space using a template as a reference. Nowadays, this process is accomplished through several tools that sometimes take several steps to achieve this goal. Furthermore, they can take a long time to achieve the results,

results that are not always up to expectations. A better solution would therefore improve the everyday work-life of researchers.

Deep Learning, DL for short is, as shown in Figure 1-1, a subset of one of the subfields of Artificial Intelligence (AI) called Machine Learning [6][7]. It enables computers to learn from past experiences and understand the world as we need [8].

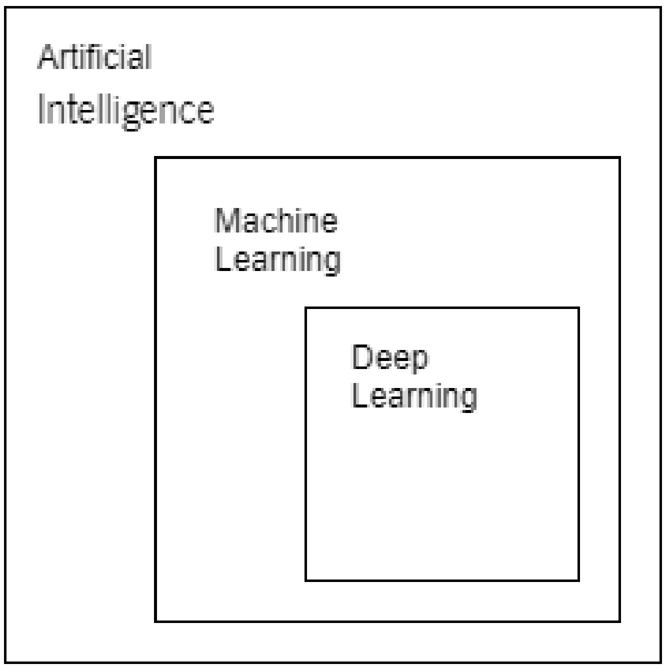


Figure 1-1. Deep Learning as part of Artificial Intelligence.

DL began with Artificial Neural Networks, which are inspired by the interconnections between the neurons of the brain. However, unlike a biological brain, where each neuron can connect to any other within a certain physical distance, these networks have layers, connections, and directions of data propagation [9]. Each neuron assigns a weight to its inputs and, depending on the weighting and the input data, a certain output is obtained at each neuron. To optimize the weighting, all the network needs is training, it needs to see hundreds of thousands, even millions of images so that the weightings are tuned to perfection. At that point, the network will get the answer right practically every time [9].

Deep Learning has already proven to be a fundamental tool in Medical imaging [10]. Many studies mention its benefits when used in this area. It is shown that DL can achieve faster and more accurate results than non-DL algorithms/methods when using an adequate model.

After considering the disadvantages of the existing tools and the positive aspects of DL, a model using DL could theoretically help the normalization process, both in terms of accuracy and time it takes to complete. Hopefully, this model would not only be faster and more accurate than other solutions, but also an automated or at least semi-automated solution that requires little to no human intervention when a brain needs to be normalized for a study.

1.2 OBJECTIVES

The main goal of this work was to develop a Deep Artificial Neural Network to perform the process of brain normalization faster and more accurately than other already existing solutions like the FSL tools and as much automated as possible. This way, researchers save time to focus on other more demanding tasks when they need to normalize a series of brain images for their studies. This Artificial Neural Network could then be used alongside a UI (User Interface) to perform the normalizations or integrated in tools like the XNAT server to achieve an all-in-one solution where storage and processing of the studies is made.

While studying the MRI studies, a second goal appeared for this work, regarding analysis. The second goal would be the development of another other Deep Artificial Neural Network. This time, brain MRI studies should be classified depending on their condition of cerebral atrophy. This would save time as the classification could, this way, be done automatically.

1.3 INVESTIGATION METHODOLOGY

Before the writing of this dissertation, a detailed investigation to the literature was needed to find possible approaches to solve the problem in hands. To optimize this investigation a strategy had to be used. As such the Design Science Research (DSR) was chosen as a strategy. Well known to be effective in the area of computer science and medical informatics, the DSR methodology (Figure 1-2), is a rigorous scientific research methodology that can be divided into six main steps [11][12].

Following the DSR methodology, after identifying the problem and motivation and defining the objectives to this work, literature was searched to help in the design and development of the solution to the problem in hands. To find literature, tools like Google Scholar, PubMed and others were used to ensure the quality of the information. Among the searched keywords were MRI, brain spatial normalization, Deep Learning and brain analysis.

Deep Learning was used, due to its impressive results that surpass other Machine Learning algorithms and, according to the literature, even exceed the humans in accuracy [7]. After finding some results, a good approach according to the literature, with Deep Learning would be to use a U-Net based architecture due to its impressive results in the area of medical imaging [13]. Hence, U-Nets were investigated to obtain information on how to implement one, starting the first experiment.

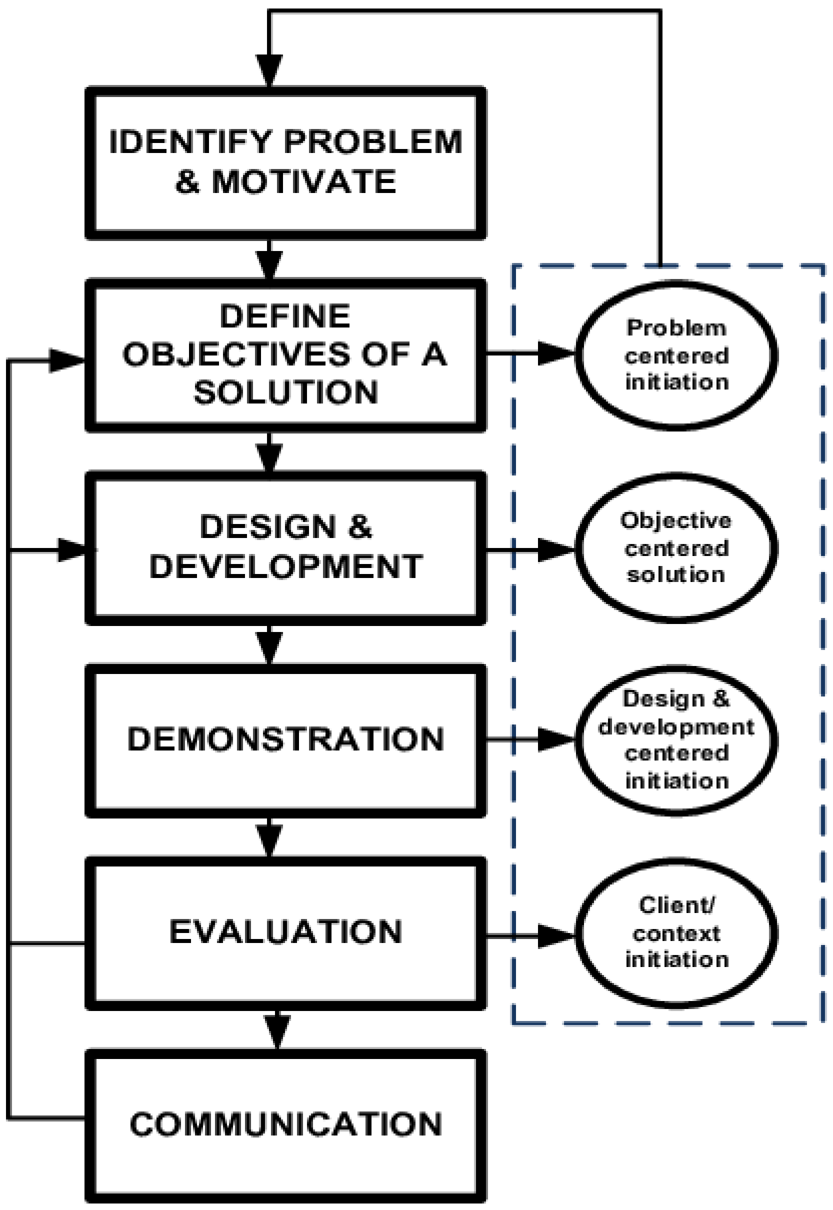

IDENTIFY PROBLEM & MOTIVATE
DEFINE OBJECTIVES OF A SOLUTION
DESIGN & DEVELOPMENT
DEMONSTRATION
EVALUATION
COMMUNICATION
Problem centered initiation
Objective centered solution
Design & development centered initiation
Client/ context initiation

Figure 1-2. DSR process diagram [14].

1.4 STRUCTURE OF THE DISSERTATION

This dissertation is structured in 5 sections. First, an Introduction is given to bring the reader into contact with the problem and to specify the objectives of this dissertation. The next section, Technologies and Concepts, describes in more detail the problem as well as the technologies used throughout this work to overcome it. The section Spatial Normalization of the Brain then describes the materials and methods for solving the main problem described in the first sections. The solutions achieved are also included in this section. Subsequently, the Brain Atrophy Classification section describes the solution for the secondary problem as well as the results obtained. Finally, this dissertation ends with Conclusions summarizing everything that has been discussed and presenting what can be done in a future work to improve the outcome.

2 Technologies and Concepts

This second section describes all the technologies and concepts used throughout this work and their use. It also describes the process of the spatial normalization and tools involved in that process to better contextualize the reader on how a better solution would improve the researches job.

2.1 MAGNETIC RESONANCE IMAGING

Magnetic Resonance (MR) is an amazing combination of advanced science and engineering, as this technology makes use of fields like superconductivity and quantum physics to obtain a useful image of a part of a patient's body [15]. Along the years, Magnetic Resonance Imaging (MRI) has evolved from unpromising beginnings in its early stages to becoming nowadays the imaging method of choice for a large proportion of radiological examinations [15]. One of the reasons for being the modality of choice is its relative safety as it is 'noninvasive' [16]. It uses radiation in the RF range, unlike X-rays, which doesn't damage tissue and utilizes the natural magnetic properties of the human body to produce detailed images [17]. This makes it also safer than other modalities since X-rays have long been known to increase the risk of cancer [18]. An MRI scanner is represented below in the Figure 2-1.

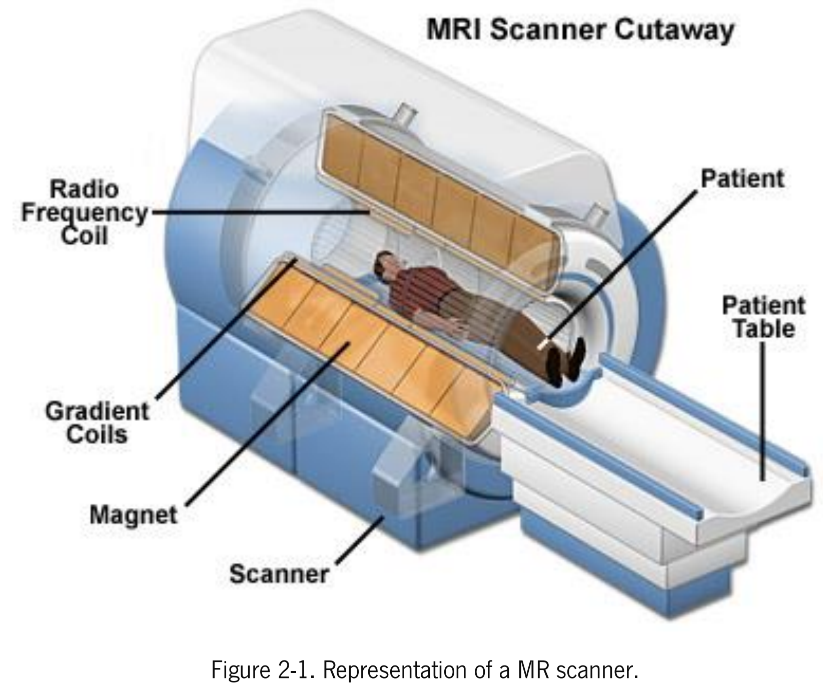


Figure 2-1. Representation of a MR scanner.

Even though the physics behind how this modality works are too extensive and complex to cover in this dissertation, some basis are required to have a better understanding on this matter.

Essentially this modality makes use of the water present in the human body, more specifically the magnetic properties of the hydrogen nuclei which are a part of the water molecule [15][16][17]. While acquiring images with this modality, essentially three phases are involved. In the first one, the magnetic phase, the MRI scanner emits energy to the targets' body in the form of a magnetic field. This magnetic field (B_0) aligns the hydrogen nuclei in the body from a random spin to a spin with the same direction as the magnetic field created by the scanner. A greater proportion of nuclei aligns parallel to the magnetic field (low energy nuclei) than antiparallel (high energy nuclei) as seen on Figure 2-2.

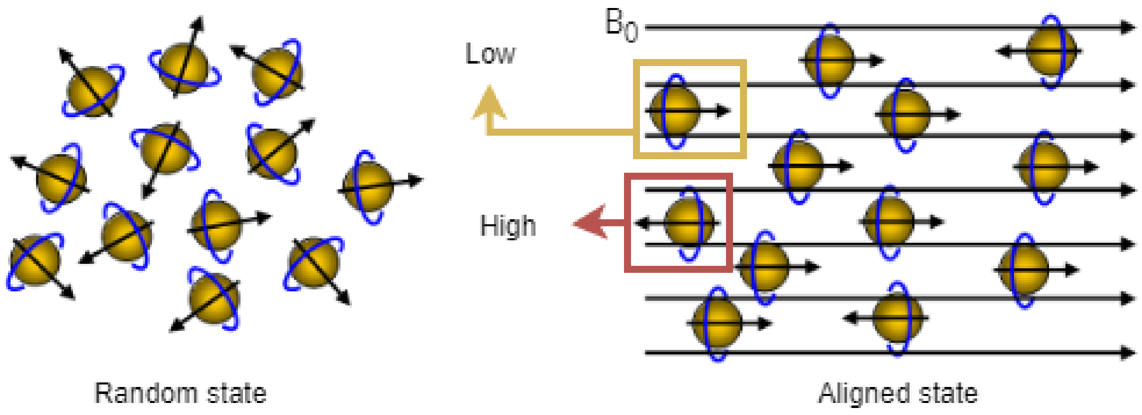


Figure 2-2. Spin alignment to the magnetic field.

After this phase of alignment enters the second phase, the resonance. In this phase Radio Frequency (RF) pulses are applied to the, now aligned, nuclei to cause disturbance in the alignment [16][17]. It does so by emitting RF pulses at specific frequencies that resonate with some nuclei. Several frequencies are needed as the nuclei don't resonate with the same frequency. The scanner can also emit frequencies just in certain areas where an image is to be obtained. The time to reach equilibrium after this disturbance is known as relaxation time. And finally comes the last phase, the imaging. The relaxation is different from nuclei to nuclei, however, all the nuclei (hydrogen nuclei in most of the cases) disturbed by the resonance of the RF pulses absorb energy in that stage and, in the relaxation release this energy. This energy is then captured by the scanner among other information and then the image is constructed in a computer.

Several types of MRI exist such as the discussed in the next sub sections: structural MRI, functional MRI and DTI. But other exist like FLAIR (Fluid-Attenuated Inversion Recovery) not depicted in this dissertation.

2.1.1 STRUCTURAL MRI

Structural MRI as the name implies, essentially involves the MRI studies regarding the structure of the human body. Several contrasts can be applied, noting that changing the contrast is not the same as changing the window or level of the image [15]. Varying this contrast or the timing of the sequence can for instance make a tumor seem more apparent among the brain tissue by making the tumor darker and the brain tissue lighter [15].

As said before, the Relaxation times describe how long the tissue takes to get back to equilibrium after the RF pulses. The well-known relaxation times are the spin-lattice relaxation time and spin-spin relaxation time, denoted T1 and T2 respectively [15]. These two times depend on the tissue as described in the Table 2-1.

Table 2-1. T1 and T2 times (in milliseconds) for different tissues [15].

Tissue type	T1 times	T2 times
Fluids	1500–2000	700–1200
water-based tissues	400–1200	40–200
fat-based tissues	100–150	10–100

As it can be seen on the table, T2 is always shorter that the respective T1 for each tissue type. The T1 and T2 (among other variables) influence the image contrast. For example, T2-weighted exams show tissues with long T2 with a bright appearance. T1-weighted exams on the other hand are the opposite, bright pixels on T1 are associated with short T1s [15]. The Table 2-2 shows a summary of how the tissues are represented in T1 and T2 MRI.

Table 2-2. Summary of T1, T2 sequences [15].

Sequence	Recognition		Good for
T1	Tissue	Contrast	Known as 'anatomy scans': Boundaries between different tissues are clearer
	Fat based	Bright	
	Fluids	Very Dark	
	Water based	Grey	
T2	Tissue	Contrast	Known as 'pathology' scans: Abnormal fluids are bright Normal tissue is darker
	Fat based	Grey	
	Fluids	Bright	
	Water based	Grey	

In the brain, MRI techniques like structural MRI can differentiate between White Matter (WM) and Grey Matter (GM). These techniques can also detect aneurysms and tumors by differentiating them from the "normal" brain tissue.

2.1.2 FUNCTIONAL MRI

Functional Magnetic Resonance Imaging (fMRI) is a type of MRI used to image changes in the neural activity by checking changes in neural metabolism [19]. We do so by monitoring oxygenation changes in the tissues [15]. These changes can be triggered by asking the patient to perform certain tasks, in order to target a specific cognitive process, or can occur naturally while the patient is in "resting state" [19]. For example, if a volunteer is asked to move a thumb, the signal received by the scanner would show an increase in the flow on a specific motor area of the brain [16]. The fMRI signal intensity variation is a result of the BOLD, Blood Oxygen Level Dependent which depends on the ratio of oxyhemoglobin which is diamagnetic and deoxyhemoglobin which is paramagnetic [19][20]. This signal intensity doesn't change much but, by modulating the oxygenation in the areas in study using the tasks, a difference between the images can be made and the area where the oxygen was more concentrated becomes evident. We do this by subtracting the control image (image where the oxygen was equally distributed by the brain) to the image acquired when the task is being performed, the experimental image. This process is illustrated in Figure 2-3.

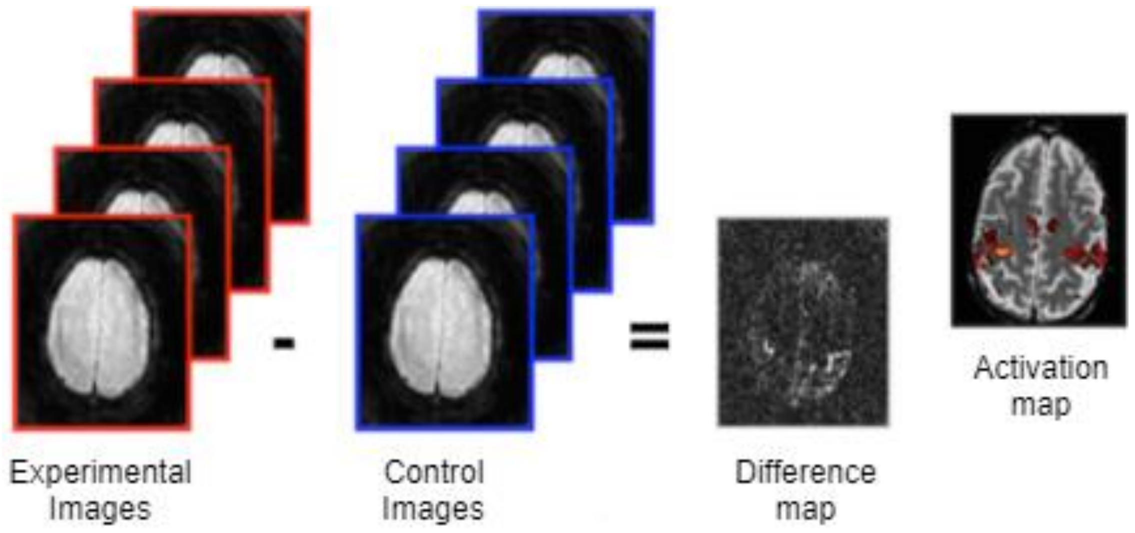


Figure 2-3. Illustration of fMRI process – based on [19].

2.1.3 DIFFUSION TENSOR IMAGING MRI

Diffusion tensor imaging (DTI) MRI is a type of MRI that consists in a group of techniques where various diffusion properties of a certain tissue can be analyzed and are used to create images [21]. It is nowadays well established that the MR diffusion tensor can provide useful information of the body that is not available from other imaging modalities [22]. Diffusion is the process by which matter is irreversibly transported as a result of random molecular motion [23].

The sensitivity of MR signals to diffusion has been known since over half a century ago. However, only more recently has diffusion-weighted MRI (DW-MRI) established itself as an important method [15]. This type of imaging behaves to some extent like an inverse of T2 weighting imaging described earlier. In this case, water-based tissues give lower signal intensity whilst other more solid tissues give a stronger signal. The possible orientation of the water molecules in a tissue in study in combination with the applied RF gradient direction will also determine the obtained signal intensity. Generally, the diffusion properties are described mathematically by a diffusion tensor which is a matrix of nine values, each corresponding to a gradient orientation and a cell orientation [15][23]. The diffusion tensor is defined by:

$$DT = \begin{bmatrix} D_{x,x} & D_{x,y} & D_{x,z} \\ D_{y,x} & D_{y,y} & D_{y,z} \\ D_{z,x} & D_{z,y} & D_{Z,z} \end{bmatrix} \qquad (1)$$

Noting that the tensor presents some redundancy where, for instance, $D_{x,y} = D_{y,x}$.

Using the tensor, several values can be calculated, such as values called eigenvalues and eigenvectors which are used to create the images [21]. The trace diffusion constant can also be computed from the values in the tensor using the expression:

$$Trace(D) = D_{x,x} + D_{y,y} + D_{Z,z} \qquad (2)$$

From which the ADC can then be obtained with:

$$ADC = \frac{1}{3} * Trace(D) \qquad (3)$$

The ADC, apparent diffusion coefficient, is usually sufficient to characterize the diffusion properties of a tissue [22].

Table 2-3. Common Apparent Diffusion Coefficient (ADC) values for human brain elements [15].

Tissue	ADC value (x10^{-3}mm^{2}s^{-1})
CSF	2.94
Grey Matter	0.76
White Matter	0.45

2.2 SPATIAL NORMALIZATION

Sometimes the fMRI data is obtained only with the purpose of understanding just one person. This data can then be used, for instance, to plan a surgery to remove a tumor on that person.
Most of the times, however, a generalization across individuals is needed so that it is possible to take conclusions about the brain's function or structure that apply to conditions or even species more broadly. To make this possible, the data needs to be integrated across individuals. This constitutes a problem as individual brains greatly differ in size, shape and positioning within the scanner. As such, it is necessary that the brains are firstly transformed so that they are aligned with each other.

The process of transforming the brains to align them in the same space is known as spatial normalization [24].

2.2.1 BRAIN ANATOMY

The brain is a part of the central nervous system. Its anatomy is very complex as can be seen in the Figure 2-4, but it can be divided in three main parts. These main parts are the forebrain (cerebrum and diencephalon), midbrain and hindbrain (pons, medullaoblongata and cerebellum) [25]. The largest part of the brain is the cerebrum which consists of two cerebral hemispheres which are connected by a mass called the corpus callosum [25].

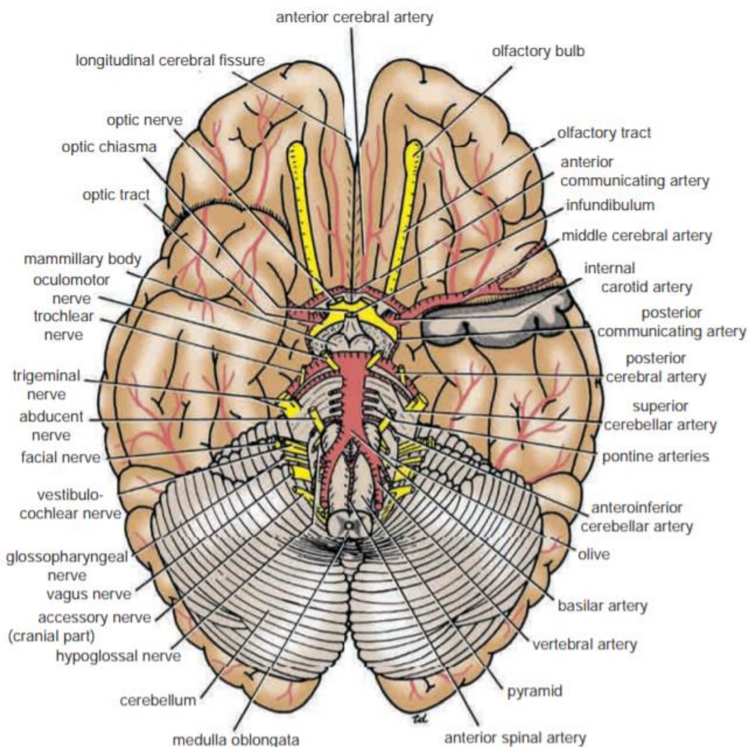


Figure 2-4. Arteries and cranial nerves on the inferior surface of the brain [25].

2.2.2 BRAIN TEMPLATES

To align different brains in a single space, we first need a reference space to align them into. For that there are two main template systems that are widely used for most MRI studies, the Talairach Atlas (which is the oldest of the two) and the MNI templates [24][26]. Both templates represent a reference image of what a normalized brain should look. Having that reference allows the tools to compute the brain exams to perform the spatial normalization.

2.2.2.1 TALAIRACH ATLAS

The Talairach Atlas, represented in Figure 2-5, is one of the best-known brain atlases. Created by Talairach in 1967 and afterward updated in 1988 by Talairach & Tournoux, this atlas provided a set of sagittal, coronal, and axial sections that were labelled. It presented three important innovations: a brain coordinate system (Talairach coordinate system), a spatial transform (to match different sized brains) and an atlas of a brain oriented according to the defined coordinate system [27]. However, this atlas presents several problems. A major one is that there is no MRI scan available from the individual on whom the atlas is based upon and, therefore, an accurate MRI template cannot be created for this atlas [24]. This happens because the brain represented in the atlas is from a, at the time, 60 year old female postmortem specimen, whose brain was sliced sagittally and used to create the drawings of the atlas [27].

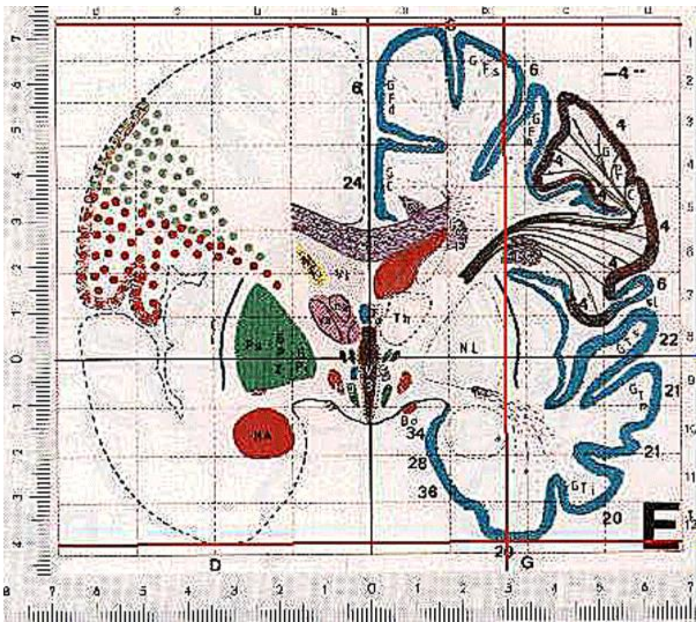


Figure 2-5. The Talairach Atlas [26].

2.2.2.2 MNI TEMPLATES

The most common templates used nowadays for spatial normalization are those known as the MNI templates, represented in Figure 2-6, developed at the Montreal Neurological Institute. They are commonly referred as MNI-152 as the template represents an average of 152 different brain scans [27]. These templates were developed to provide an MRI-based template that would allow automated registration rather than landmark-based registration as in the Talairach Atlas [24]. Unlike the atlas described before, these templates provide an MRI template, in NIfTI format to be used by the tools.

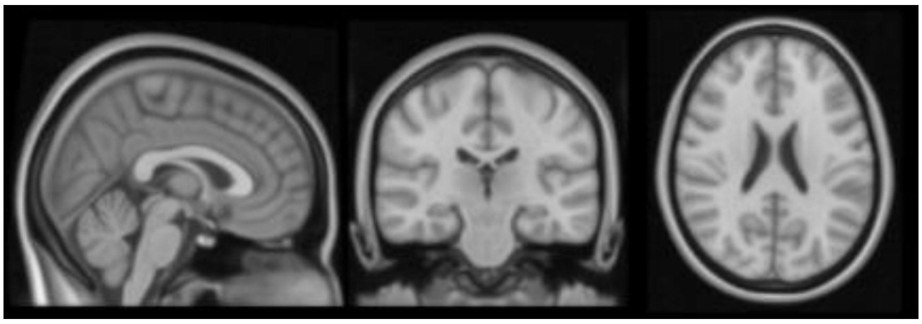


Figure 2-6. The MNI-152 Template.

2.2.3 TOOLS

Typically, in neuroimaging, to obtain answers we need to extract pertinent information from imperfect images of the brain [28]. When dealing with MRI, several tools are needed to obtain these desired answers. Sometimes a normalization of the brain's MRI is required to correctly obtain information and, as such, tools to perform such transformations are needed. Several programs can perform this process, such as FSL, which will be the main program used and described throughout this thesis, but also SPM, ANTS and Brain Suite.

Regarding FSL, it is a comprehensive library of analysis tools for fMRI, MRI and DTI brain imaging data [29][30]. This library contains tools such as bet, Flirt and Fnirt that are used in the normalization process.

The bet tool, Brain Extraction Tool, uses a deformable model that evolves to fit the brain's surface. It then removes the non-brain part of the image, leaving just the brain tissue. It's a very fast method and requires no pre-processing [31][32].

The Flirt tool, FMRIB's Linear Image Registration Tool, is a fully automated tool for linear brain image registration, both intra- and inter-modal [33]. This tool has been used by several researchers including trained neurologists, psychologists and physiologists to perform thousands of registrations in the context of fMRI analysis and structural studies [34]. As Flirt is a linear tool, it can translate, rotate, zoom and shear one image to match it with another [35].

The Fnirt tool, FMRIB's Non-linear Image Registration Tool, is similar to Flirt but unlike it in many ways, as it is a non-linear tool. The local deformations permitted by Fnirt, as it is a non-linear method, may accomplish better results when the differences between subjects are such that the linear transform (performed by Flirt) is not sufficient to achieve good registration [35].

All the tools described before can be seen in Figure 2-8, which explains the process of spatial normalization and clearly shows the inputs/outputs of each tool.

2.2.4 PROCESS

Although the tool used to normalize the brain images works, sometimes the results are not perfectly accurate.

The Figure 2-7 shows an example of a normalization comparing the original image, the MNI template (Reference) and the output which has a small error marked red. In this case, the error is not significative as it doesn't affect the brain tissue in the image.

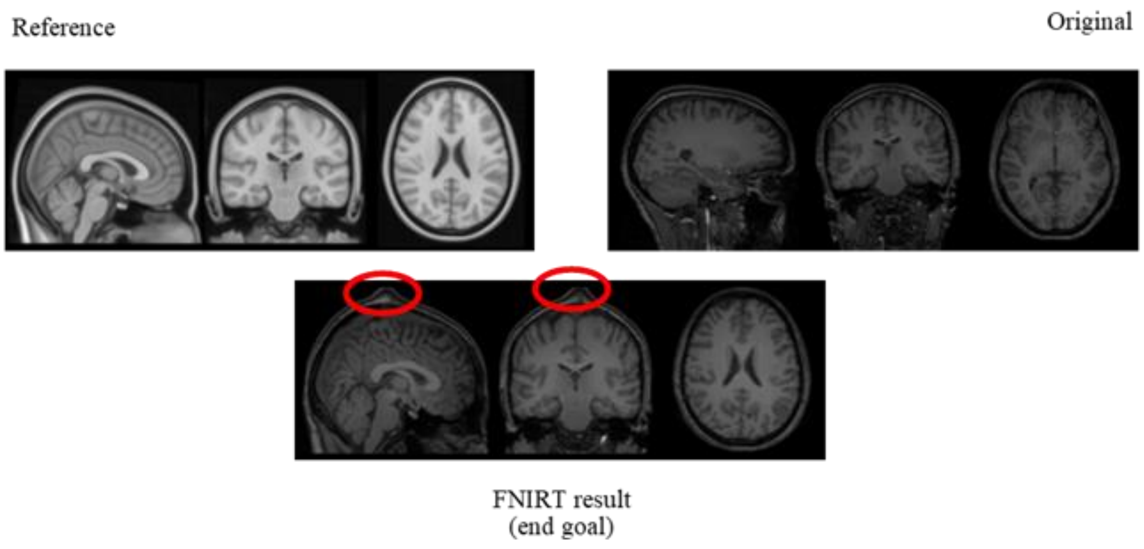


Figure 2-7. Comparison between original, reference and resulting image with a small error marked in red.

The process to normalize the images using FSL is illustrated in Figure 2-8, the figure also shows a side-by-side comparison of the original and final image with representations of intermediate steps.

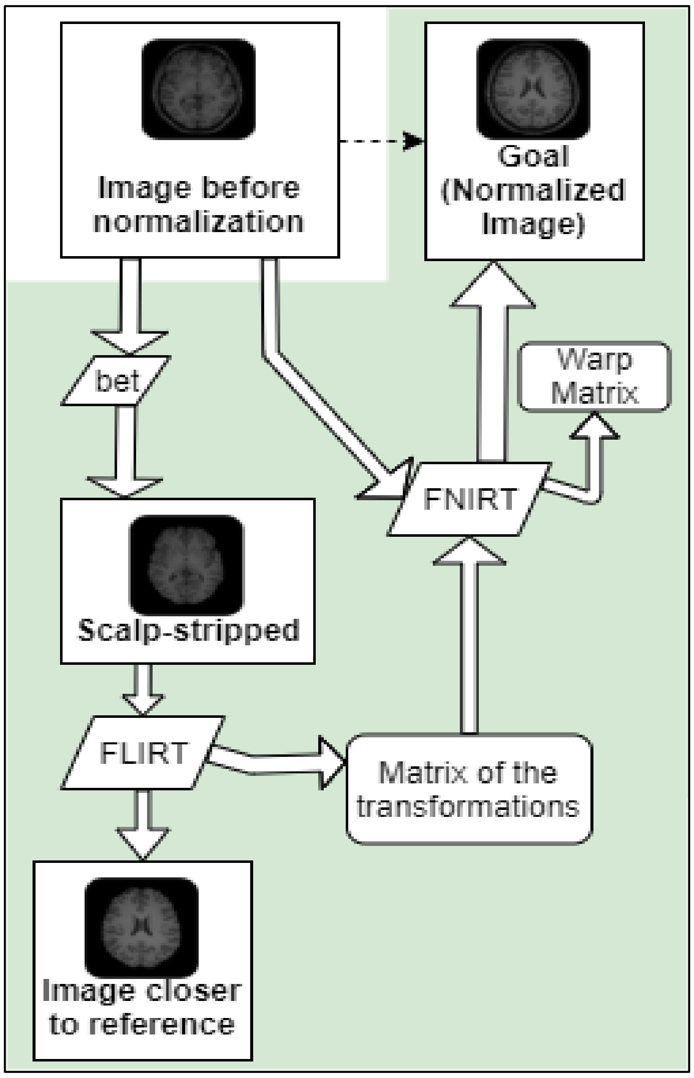


Image before normalization
Goal (Normalized Image)
bet
Warp Matrix
FNIRT
Scalp-stripped
FLIRT
Matrix of the transformations
Image closer to reference

Figure 2-8. MNI Brain Normalization Process Diagram.

As represented in Figure 2-8, the process for brain normalization using FSL tools is as follows: Firstly, the raw data coming from the machine may have undergo pre-processing to possibly correct errors that may happen when acquiring the image. This pre-processing, although not mandatory, may include movement correction for instance. After that, the original image i.e. the image before the normalization process is obtained. In the next step, the brain is extracted from the image, using the bet tool. To the resulting image is then applied the Flirt tool, which makes linear corrections to the image of only the brain to approximate it from the reference image, giving two outputs, an image of the extracted brain (which is now closer to the reference) and a text file which describes the changes made to achieve the result from Flirt. Finally, the Fnirt tool is used, it uses the original image, before the brain extracted, the matrix produced from Flirt and, again, the standard image.

This tool, opposite to the previous one, distorts the image in a non-linear way, i.e. expands or contracts several spots in the image instead of the linear version that stretches or contracts the image equally in the whole axis or even rotates the image. The result from Fnirt is the so desired normalized image as well as a special matrix (designated warp matrix) which contains the information of the local deformation for each of the individual voxels of the original image to obtain the final result. The final image can then be used by the researcher as a normalized version of the initial image and can be saved in a database or archive like, for example, XNAT.

2.3 XNAT - NEUROIMAGING ARCHIVE

When dealing with neuroimaging, a lot of data is generated, data that needs to be handled, either to be stored securely or to be distributed. This can sometimes be a challenge so, to make it easier to handle all the data XNAT can be used. XNAT (Extensible Neuroimaging Archive Toolkit) is an open source imaging informatics software platform [36]. XNAT was developed to facilitate common management and handling of neuroimaging and associated data [37]. It can handle importing, archiving, processing and securely distributing of that imaging and its related study data. As can be seen on Figure 2-9, the XNAT server is able to capture data from multiple sources, maintain it in a secure repository, and then distribute it, when needed, to authorized users. The user access to the archive is done by a secure web application which provides several quality control and productivity features, including detailed reports, upload and download tools as well as security tools [37].

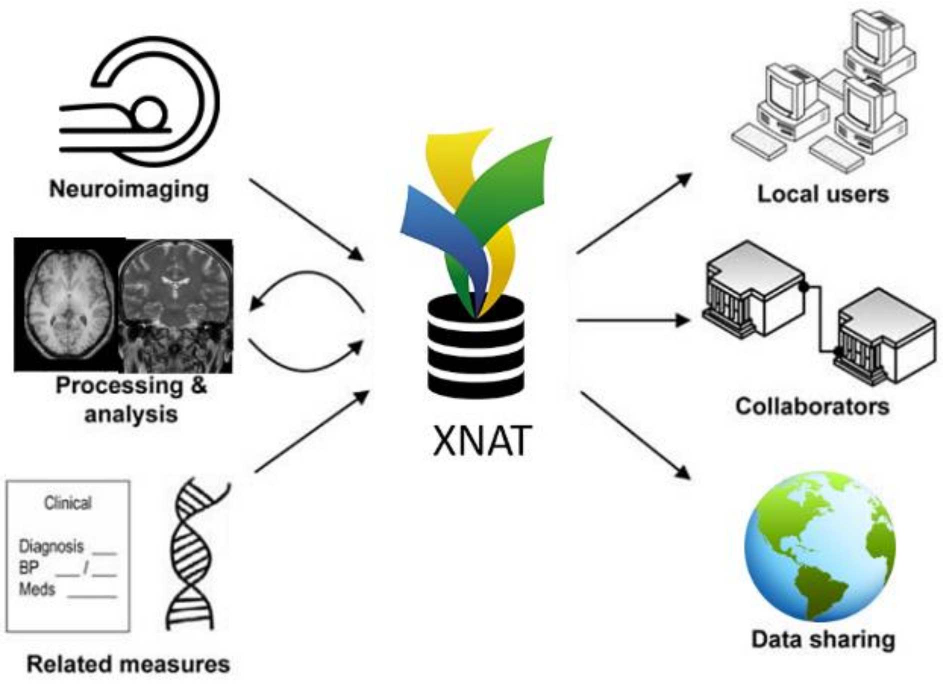


Figure 2-9. XNAT captures data and distributes it securely to a variety of end users - based on [37].

The server can also be accessed by using one of several python libraries capable of bypassing the web interface and interact directly with the server. One of these libraries is PyXNAT, originally developed in the context of the IMAGEN European project, PyXNAT acts as an interface to an XNAT database [38]. Other library capable of handling XNAT via python is xnatum[1] [39]. This last library was the one chosen to be used in the container.

A workflow is implemented by XNAT to support the quality, integrity, and security of data. Imaging data from the scanners enter the workflow using mechanisms like DICOM (Digital Imaging and Communications in Medicine) "pushes", SFTP (Secure File Transfer Protocol), or portable hard media. Non-imaging data, on the other hand, such as clinical assessments, subject demographics and genetic measures are passed via web-based forms, spreadsheet uploads, or XML (eXtensible Markup Language) [40]. Data can only be stored within XNAT if organized according to projects. This required project association is the basis of the XNAT security model as users are only given access to data which belongs to a project they are included [41]. To keep data organized and tailored to the user needs, projects are just one of the many hierarchy levels of the XNAT data structure. Inside a project, several subjects can be inserted, where a subject is anyone participating in that specific project associated study. These subjects don't exist outside the project they are inserted into. The

[1] Currently in version 1.2

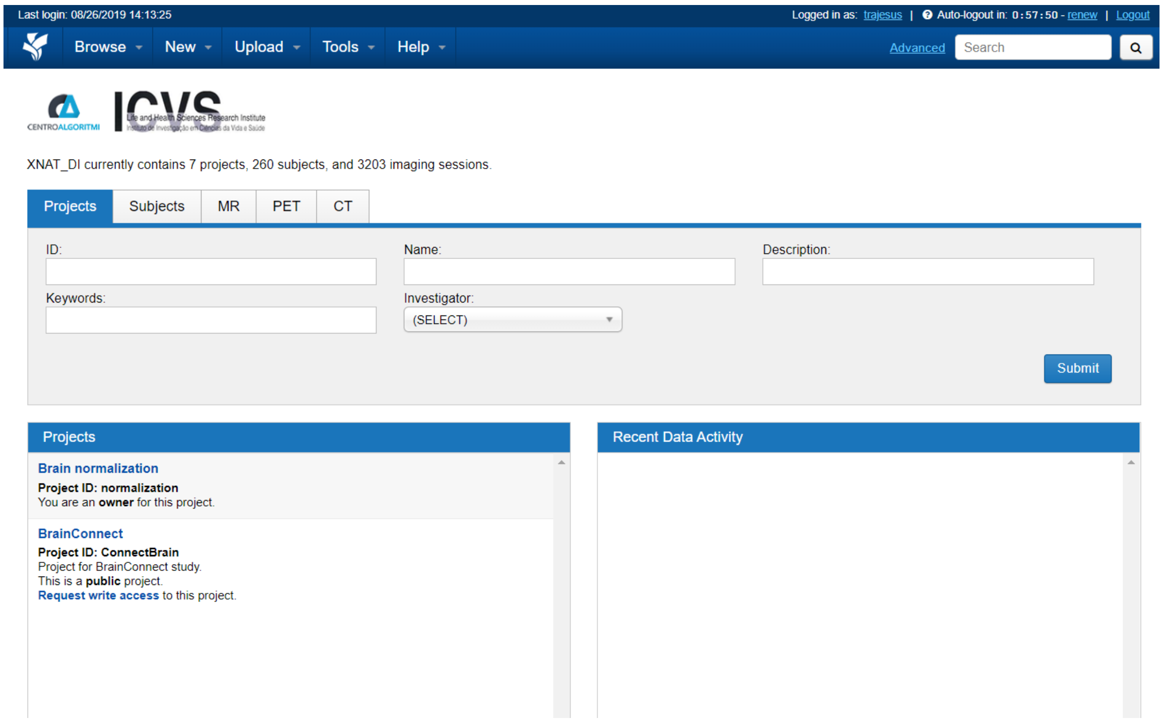
fields of information to be filled for each subject can be defined before creating the subjects and this information can be as short as just a single identification number or as extensive as the user needs it to be, including fields like the gender of the subject or the age for instance. By creating and assigning at least one subject to a project, the next hierarchy level can be accessed. A subject can contain multiple experiments which is the term used in XNAT to describe data gathered or measured from direct interaction with a subject. The data inside these experiments ranges from non-image data like a consent form signed by the subject to image data like, for instance, MR or PET sessions acquired in a scanner.

The experiment level is then divided in 3 sub-levels: scans, reconstructions and assessments. The first one, as the name implies, represents the individual acquisitions, or Series in DICOM terminology, at the scanner. The second sub-level, reconstructions, represents the files that join data from multiple scans like, for example, an image resulting from an average across multiple runs. This sub-level is intended to store simple imaging data that rather than the complex data structures other levels store. The final experiments sub-level, assessments, represents the derived data generated from processing and analyzing image data such as images, logs and spreadsheets. In addition to these levels and sub-levels, each hierarchy level has its own child level called resources.

XNAT uses a special file, an XSD (XML Schema Document), to define its core data model, including all the hierarchy levels. In that file, projects, subjects and experiments are defined as the three default core data structures. All other necessary structures to be used can be defined and added by relating to one of those core types. This highlights one of the main advantages of XNAT which is, as the tool's name implies, the extensibility as the XNAT allows its users to extend the schema to create new data types as needed. To create a new data type the user just needs to create the XSD that defines the data type itself [41].

By using the newest XNAT 1.7 version, new functionalities such as these custom data types can be added via plugins, self-contained packages that are separate from the XNAT server but, once added and enabled, run as a fully integrated extension to the XNAT server core.

As explained before, XNAT provides a web interface to access the data. Figure 2-10 shows the interface used in this work. In that figure, several key features of XNAT explained before can be seen. The projects tab lists the projects the user has access to see and projects with write permission. In the figure, only the Brain normalization project has full access. The subjects tab can also be seen.



Last login: 08/26/2019 14:13:25

Logged in as: trajesus | Auto-logout in: 0:57:50 - renew | Logout

XNAT_DI currently contains 7 projects, 260 subjects, and 3203 imaging sessions.

ID:

Name:

Description:

Keywords:

Investigator: (SELECT)

Projects

Brain normalization
Project ID: normalization
You are an **owner** for this project.

BrainConnect
Project ID: ConnectBrain
Project for BrainConnect study.
This is a **public** project.
Request write access to this project.

Recent Data Activity

Figure 2-10. XNAT web interface.

2.4 DEEP NEURAL NETWORKS

The human brain is one of the most important organs of our body and the most complex of them all. Due to this very important organ, humans are capable of thinking, producing and accessing memories, but most importantly, acquiring new information and learning from it. Computers on the other hand can't simply acquire information and learn from it in order to solve a given problem. However, given that usually computers can perform tasks faster than humans, developing a way for a computer to, in some way, learn to perform a task as expected should be beneficial. This is where Deep Learning comes in. Deep Learning or DL for short is, as represented in the Figure 2-11, a subset of Machine Learning, one of the subfields of Artificial Intelligence [6][7].

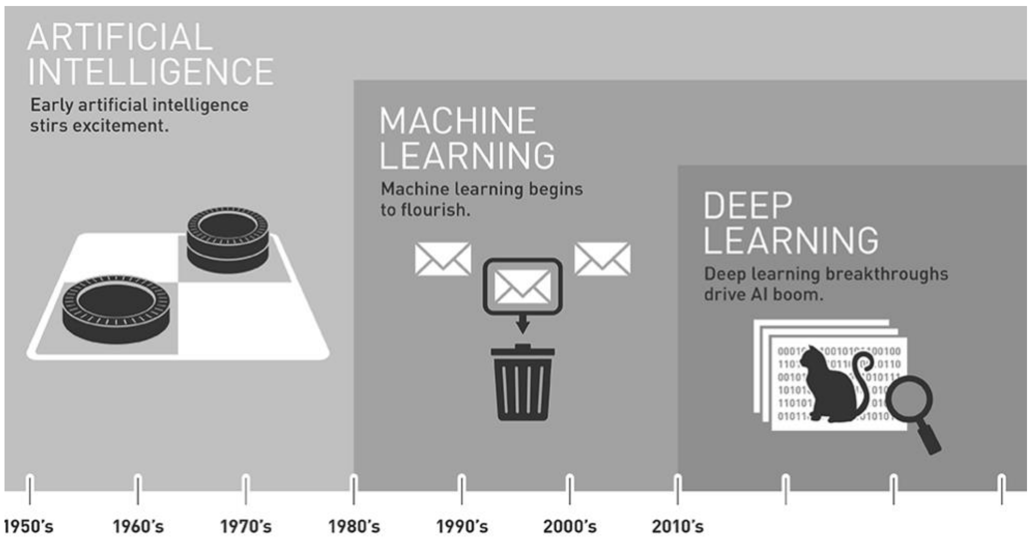

ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Figure 2-11. The difference between Artificial Intelligence, Machine Learning and Deep Learning [9].

DL had its origins in the Artificial Neural Networks which are inspired by the interconnections between the brain's neurons. Unlike the human brain however, the Artificial Neural Networks are constituted by artificial neurons, a computational version of the original natural neurons encountered on the human brain. Each of the neurons constituting the Artificial Neural Networks assign a weight to its inputs and, a certain output is obtained at each neuron depending on the input. In order to optimize the weights used in each artificial neuron, and in consequence 'learn' how to achieve a good output to that input, the network needs some training. This learning process of the network can, once again, be compared to the human learning. Taking, children as an example, while growing up, they learn to recognize different objects, animals or patterns in general. They do so by seeing lots of examples of what they are learning and, after trying to guess what it is, they are told by their parents if their answer was, or not, correct. In both scenarios, the feedback from the parents is important because if the answer was correct, they now learned to recognize that object. If on the other hand, if they were wrong, they are corrected by the parents to improve the way they recognize the object.

The process is the same whether we try to learn new patterns or a physical task such as driving a car, we learn from the mistakes and improve our knowledge. Either case, learning means getting better at a task. This is what Machine Learning focuses on, developing algorithms that try to get better at a certain task over time. As the tasks get more complex, the data involved also gets more complex. To accommodate that complexity and achieve good results, Deep Learning is used. The deep in Deep Learning is due to its immense amount of layers [10]. These Deep Learning algorithms not only far surpass other kinds of ML algorithms, but also rival the accuracies achieved by humans [7].

2.4.1 ARTIFICIAL NEURAL NETWORKS

The base unit of the human brain is the neuron, a cell optimized to receive and transmit information from and to other neurons, processing information along the way. This natural phenomenon can be translated into computers to create artificial models capable of learning.
An Artificial Neural Network (ANN) is a hierarchical composition of basic computational elements known as artificial neurons or perceptrons [10]. Very different from the natural neuron, as can be seen on the representation of the Figure 2-12, but with the same principle, the artificial neurons or perceptrons have the function of receiving information from other perceptrons, processing it in some way and then send it to the next set of perceptrons. The artificial neurons do so by taking some number of inputs x, each of which is multiplied by a specific weight w. These weighted inputs are then summed together and, in many cases, may also include a bias, which is a constant [7]. The result of this sum is then passed through an activation function f to produce the output of the neuron and be transmitted to the next neuron or neurons. The representation of the perceptron with the inputs, weights, bias and activation function can be seen on the Figure 2-13(a).

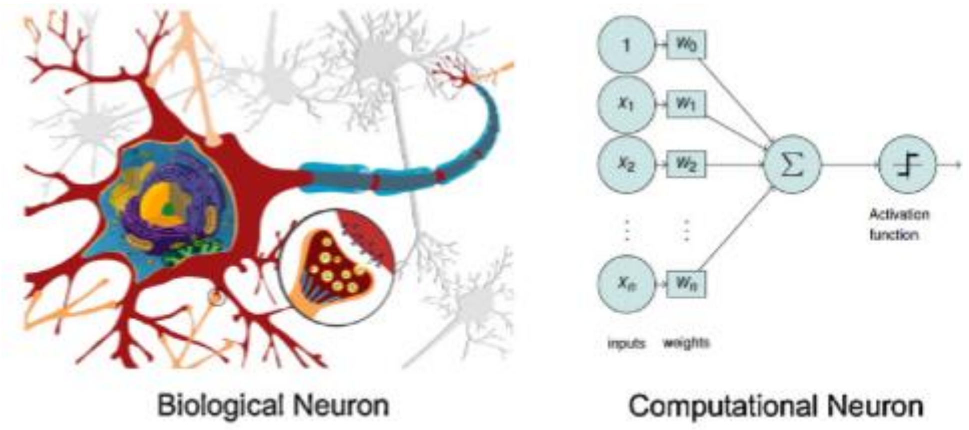


Figure 2-12. From biology to Deep Artificial Neural Networks [42].

Several of these artificial neurons can coexist in a single level of the hierarchy of the artificial model creating a single layer of that network [10]. These layers of perceptrons can then be stacked to create the ANN where the standard networks of this type are also known as Multi-Layer Perceptrons (MLPs).

The perceptrons are organized in layers because, like in the natural neuron, one unit doesn't have the power to take a decision alone, several neurons (natural or artificial) are needed to, given an input information, process it and take a conclusion. The stacking of the perceptrons can be visualized in the Figure 2-13.

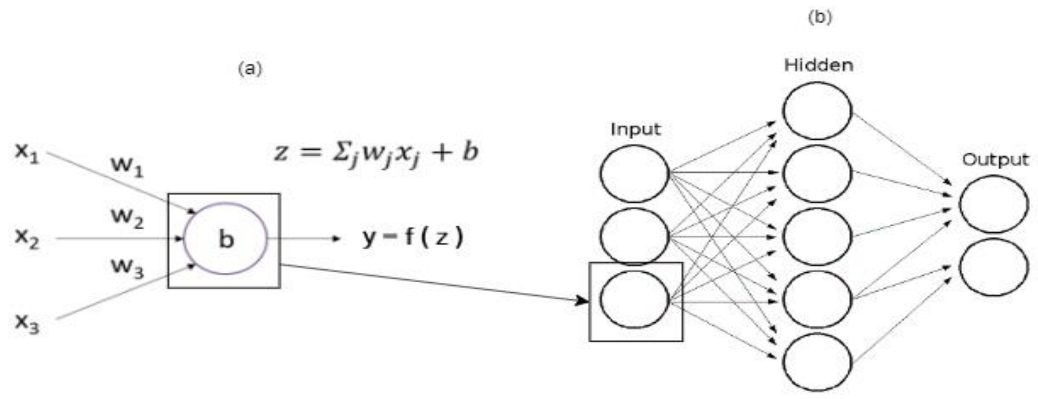


Figure 2-13. Multiple Perceptrons (a) create a Multi-Layer Perceptron (b).

An Artificial Neural Network has, typically, one layer for input, one for output and at least one hidden layer between them. Many hidden layers in an ANN make it deep [10] i.e. create a Deep Artificial Neural Network, the basis of Deep Learning.

2.4.1.1 ACTIVATION FUNCTIONS

Most neurons are defined by a function, an activation function, to be exact. That function is what defines what output will be generated from a set of inputs in a given neuron. A simplified representation of a neuron can be seen in Figure 2-14. There are several types of functions and these can be linear or nonlinear. Regarding the linear ones, the resulting neurons are easier to compute but can run into limitations [7]. On the other hand, in the nonlinear group, there are three main functions applied to neurons: Sigmoid, Tanh and ReLU.
To better understand the parameters of each of these functions further explained below and where they act in the neuron, Figure 2-14 shows a sample perceptron with all parameters labeled with three inputs as an example.

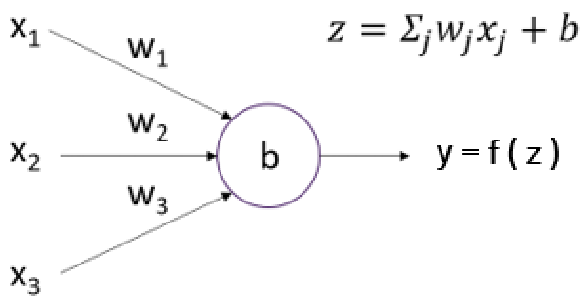


Figure 2-14. A simplified representation of a perceptron.

In the figure, x1, x2 and x3 denote the inputs; w1, w2, and w3 denote the weights and b the bias. The output y is the result of the activation function f over z.

Regarding the nonlinear functions, the sigmoid function is defined by:

$$f(z) = \frac{1}{1+e^{-z}} \quad (4)$$

As can be seen below in Figure 2-15, the Sigmoid function gives a S-shaped output where, if z is high the output y will be near the value 1 and, on the other hand, if the z value is low the output will be closer to 0.

The Tanh function, defined below, has a S-shaped graphic like the sigmoid.

$$f(z) = \tanh(z) \quad (5)$$

However, despite the same shape, they present different ranges, whereas the Sigmoid ranges between 0 and 1, the Tanh ranges from -1 and 0.

Finally, the ReLU function defined by:

$$f(z) = \max(0, z) \quad (6)$$

This function results in a characteristic hockey stick shaped output and has recently become the activation function of choice especially in computer vision [7].

Much like the ReLU, a more advanced activation function exists, the Leaky ReLU, (used in one of the models created). It behaves just like ReLU for positive values however, for the negative ones, instead of blocking them, it attenuates them. Leaky ReLU is defined by:

$$f(z) = \max(\alpha z \ , \ z) \quad (7)$$

In the function above, α denotes the attenuation to the negative values. This variable should have values in the range $0 < \alpha < 1$ where common values are 0.1 and 0.2. One particularity of this activation function is that if the value of α was 0, the function would behave like a normal ReLU and, on the other hand, if the value was 1, it would behave like the linear function.

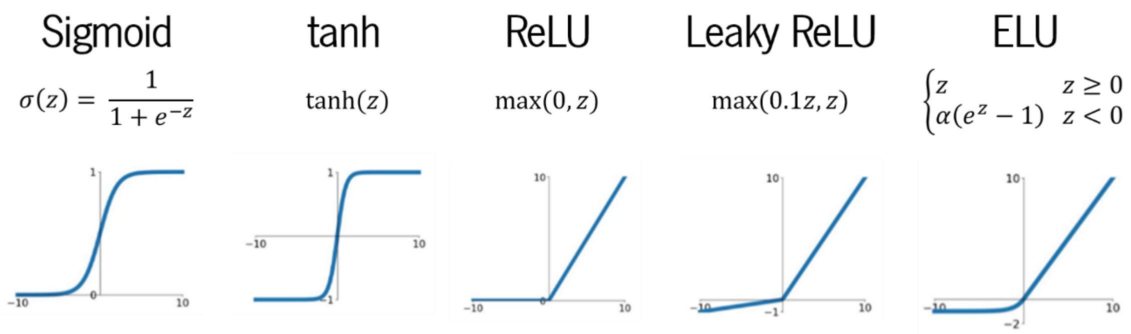


Figure 2-15. Different activation functions used in Artificial Neural Networks.

One function not represented in the image above but very useful for multi-classification tasks is the SoftMax function. It normalizes the outputs of the previous layer so that they sum up to one [6][7]. This output represents the probability, predicted by the model, of each class to be the correct output. As such, a good prediction will have one of the entries close to 1 and the rest close to 0, meaning the model is almost certain it chose the right class. On the other hand, a bad prediction will present multiple possible labels [7].

2.4.1.2 LOSS AND OPTIMIZER

Just like the activation function discussed before, other important parameters while creating a model are the loss function and the optimizer. The loss represents a penalty value for the incorrect predictions the model outputs, if no penalty was implemented, the model would not have feedback

about its output. This way, by having a penalty if the output is incorrect, the model can self-evaluate and, if all goes as expected, learn the correct features and output a correct prediction.
This penalty is then used by the learning algorithm whose task is to minimize the loss, i.e. minimize the sum of the penalties of each neuron in the network. It does so by tuning automatically the set of w (weights), b (biases) parameters on each neuron [43].
One of the most used loss functions is called cross entropy loss and is defined by:

$$C = -\frac{1}{n} \sum_{x} [y \ln a + (1 - y) \ln(1 - a) \quad (8)$$

In the expression, a denotes the application of the Activation Functions on the weights and bias, n denotes the total number of items of training data, the sum is over all x training inputs, and y denotes the corresponding desired output [43].
Analyzing the cross-entropy function, it is positive, and tends toward zero as the neuron gets better at computing the desired output of a given input [43]. This means, a zero loss would represent a perfectly correct prediction of the model on every output.
Other important loss function is the dice loss function, based on the Dice Similarity Coefficient (DSC) [44]. This loss is defined by:

$$Dice\ Loss = 1 - DSC = 1 - \ 2\frac{|X \cap Y|}{|X| + |Y|} \quad (9)$$

The Dice Similarity Coefficient is used to measure the overlap of, for instance, input and output images represented in the equation as X and Y [45]. If the overlap is perfect, the result of DSC would be 1 and, therefore, the Dice Loss tends to 0 as the overlap improves. A Dice Loss of 0 would mean a perfect overlap and consequently a model predicting correctly its output.
To minimize the penalty or loss while training the model, an optimizer is used. Some of the common optimizers are Adam (Adaptive Moment Estimation), RMSprop (Root Mean Square Propagation) and SGD (Stochastic Gradient Descent). A good explanation of how an optimization algorithm works would involve differentiable functions and statistics and is not as important to understand the main problem discussed in this dissertation, it will not be described in this work.

2.4.1.3 *CONVOLUTIONAL NEURAL NETWORKS (CNN)*

The type of architectures known as Convolutional Neural Networks are of primary importance in image analysis [10]. Like the ANN architectures, the CNNs also consist in an artificial neural network composed of multiple organized perceptron layers. However, instead of simple layers like the first, we encounter three different kinds of layers (Figure 2-16) in these networks: Convolutional layers, Pooling layers, and Fully connected layers [10].

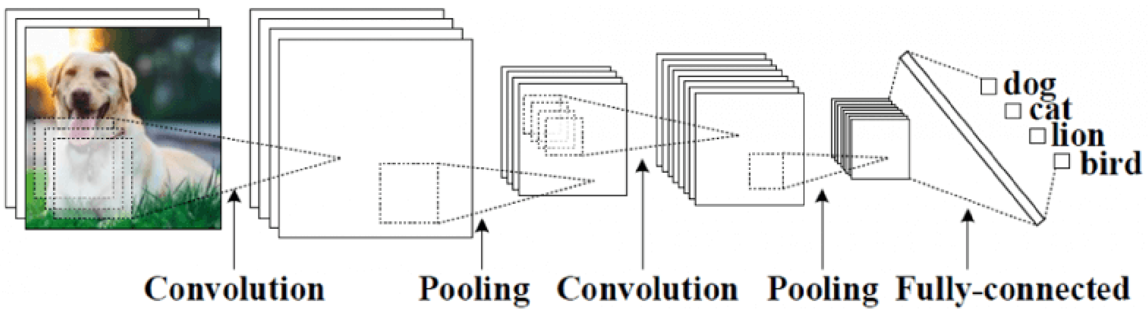


Figure 2-16. Layers of a CNN.

The first of these characteristic layers, the convolutional are the main strength of the CNNs hence the name Convolutional Neural Networks. These layers perform, as its name implies, a convolution, this means they apply several filters, or kernels, to their input image in order to extract features from that image. The result of this convolution operation (Figure 2-17) is convolved images also called feature maps as they are an extraction of the features of the initial image.

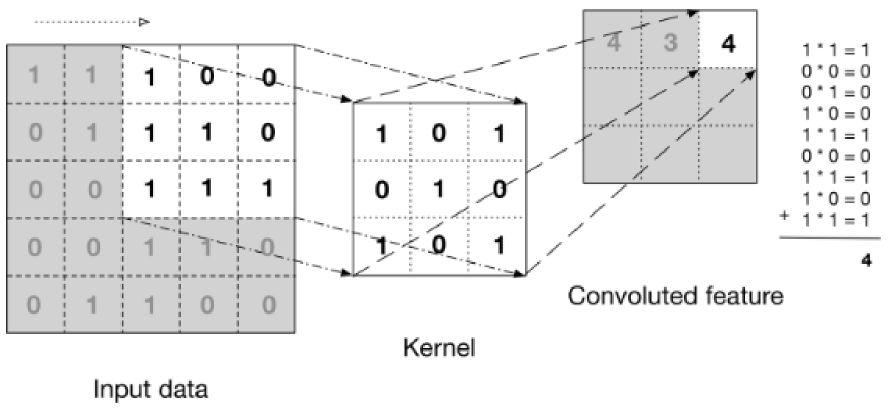


Figure 2-17. The convolution operation [46].

As shown in the figure, the convolution multiplies and sums the values of the input pixels to obtain the features.

To reduce the dimensionality of the feature maps and sharpen the features, the pooling layers are applied after the convolutional ones. By reducing the dimension of the feature maps, the pooling layers also reduce the computational complexity of the network. There are several types of pooling, one of the most common is the max pooling. It divides the feature map in tiles and afterwards runs a pooling window through them, in the example shown in the Figure 2-18, the window size is set to 2 x 2. The pooling then applies a function to the values capture by that window, in this case, the max pooling applies the maximization function which selects the maximum value of each window.

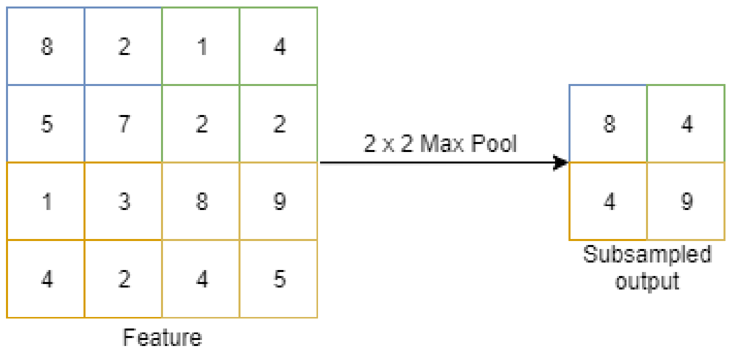


Figure 2-18. Max pooling layer.

These layers, convolution and pooling, are often seen as a CNN block and several blocks are stacked to create the CNN. The Figure 2-19 shows an example of a CNN where 2 blocks of convolution and pooling can be observed.

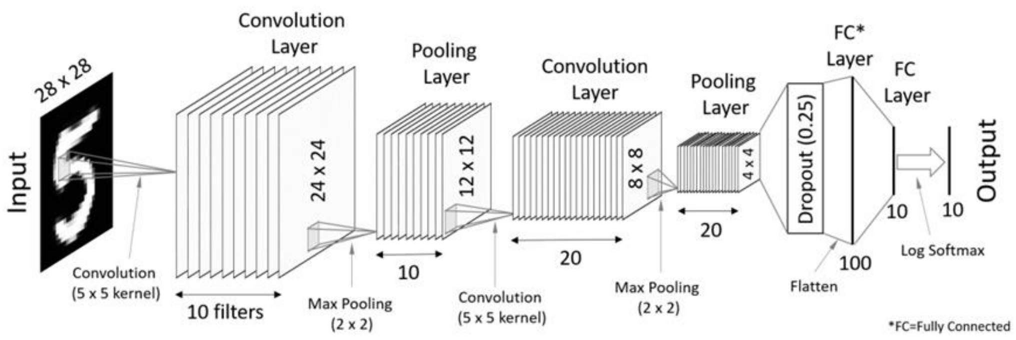


Figure 2-19. Convolutional Neural Network basic structure.

2.4.2 COMMON ARCHITECTURES

2.4.2.1 AUTOENCODERS

An autoencoder is one of the types of artificial neural network architectures. A representation of an autoencoder can be seen in the Figure 2-20. The main idea of this type of architecture is that it attempts to copy its input to the output while, simultaneously, learning a more powerful representation of that input [10]. To do so, at its core, it contains latent space consisting in a layer or a group of layers that describe the input [8]. That space is often called bottleneck as it is the narrowest part of the model in this type of architecture and being so, it only allows the characteristic features of the input to pass through, as such, if the output is as expected, it means the bottleneck can almost fully describe the input with a minimum number of features.

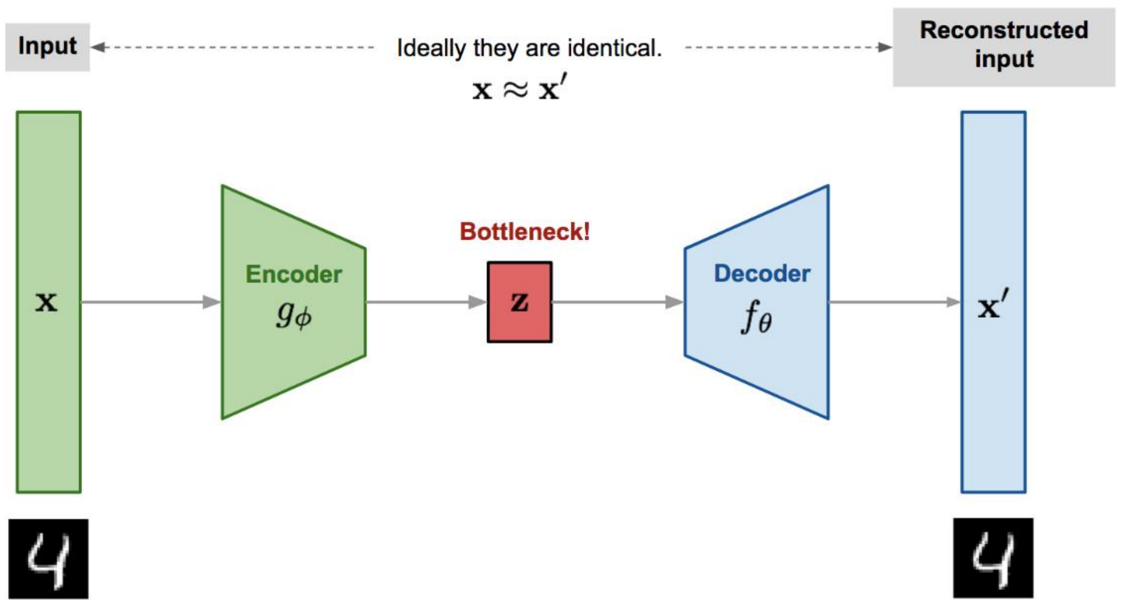


Figure 2-20. Autoencoder structure representation.

2.4.2.2 U-NET

U-net is a type of FCN (Fully convolutional network), it is known as U-Net due to its typical U shape for the architecture as represented on Figure 2-21. Like some architectures it is composed by an encoder and decoder however, unlike the others, some layers of the encoder and decoder are connected by skip connections [13]. This type of architecture has shown impressive results in the area of medical imaging, especially in image segmentation [13]. As such, due to its impressive results, this architecture was used on one of the experiments on this dissertation to attempt to solve the problem of the spatial normalization. The results of the architecture implemented can be seen in section 3.4.

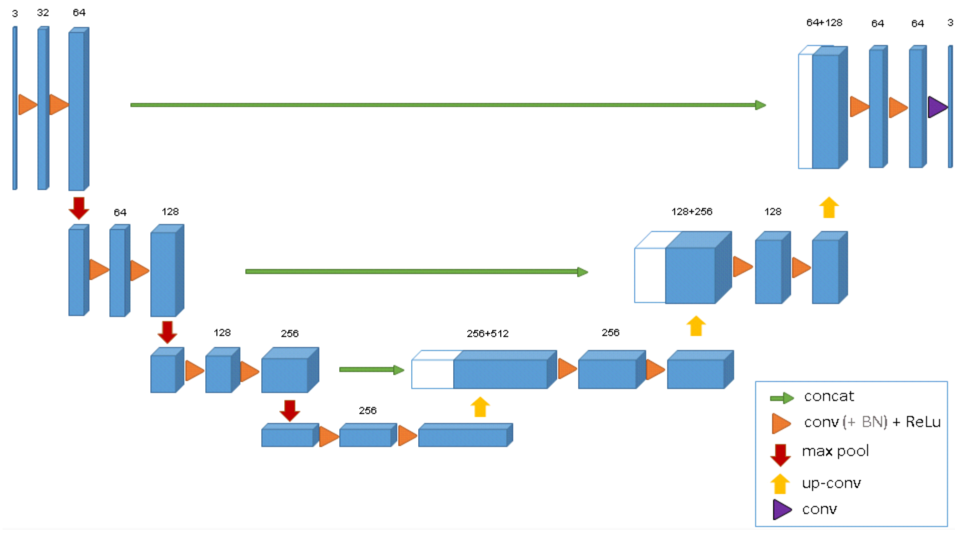


Figure 2-21. Example of a U-Net based architecture [47].

2.4.2.3 GAN

The GAN architecture (Generative Adversarial Network) is a Deep Learning Artificial Neural Network architecture proposed by Ian Goodfellow and few other researchers in 2014 [48][49].

This architecture can be considered a variation of Autoencoders. It is composed of two parts which are two Artificial Neural Networks, a generative model often called Generator and a discriminative model often called Discriminator [10][49]. The general structure of a GAN is represented on Figure 2-22. The Generator part of the GAN generates a sample which is sent to the Discriminator part as well as 'real' samples, i.e. samples of the actual training data. The Discriminator then tries to classify the sample received as real or fake [10].

The objective of the Generator is, essentially, learn to trick the Discriminator into thinking the generated samples are real. By doing so, the Generator ends learning how to generate the expected data from a given input.

This more complex architecture was used in the second experiment and the results obtained can be analyzed in section 3.5.

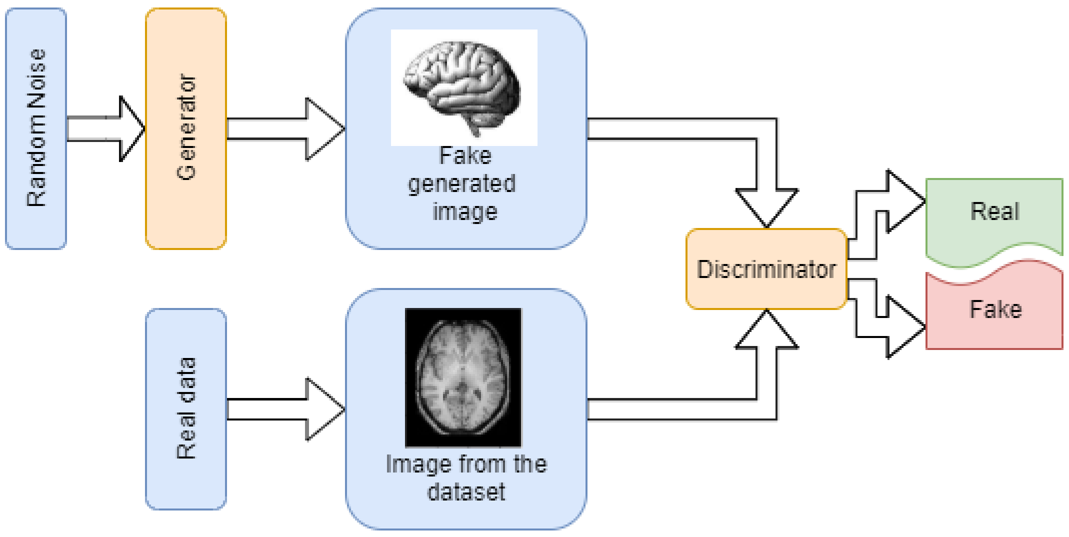


Figure 2-22. Typical GAN workflow.

2.5 WORKING ENVIRONMENT

2.5.1 FSL

FSL (FMRIB's Software Library) was developed at the Oxford Centre for Functional MRI of the Brain (FMRIB), it consists of a set of freely available software tools that can be used in brain MRI studies [50].

This library can be easily installed in several types of machines and OS. For that, the user only needs to run a script with python 2 which automatically installs all the tools within FSL and configures the path to allow the user to use these tools. The script also makes sure that the tools it installs are the latest version[2].

FSL was chosen to be used in this work due to its ease of use and powerful tools, which allow to perform the brain spatial normalization as described previously in chapter 2.2. The MRI images contained in the provided dataset were all normalized using this tool and, therefore, provide a baseline to compare the results of the model obtained.

[2] When writing this dissertation, the latest version of FSL was 6.0 released in October 2018 [29]

2.5.2 DOCKER

Docker provides a way to implement OS virtualization [51] by creating containers, minimal versions of a base OS. Through this, docker also creates an isolation between each of them, as well as with the OS itself as represented in Figure 2-23. The containers can be seen as a virtual machine, as they allow each container to have its own libraries, configuration, etc. [51]. Additionally, Docker allows each container to access common resources in the machine, specifically the CPU, GPU and RAM.

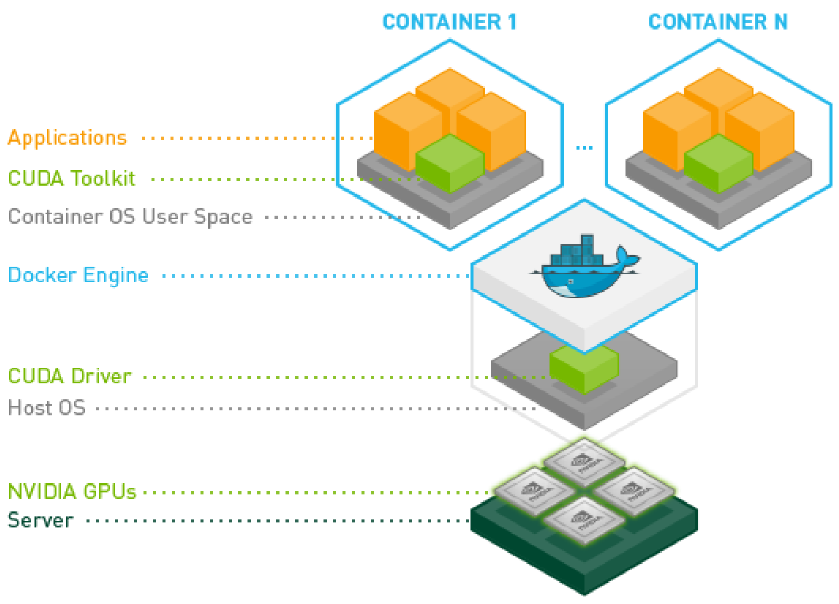


Figure 2-23. Docker containers encapsulation of applications' dependencies [52].

This makes it possible for each container to have its own experiments without compromising the machine they are running on. To start a new container, Docker builds software images by reading a Docker File, which contains the instructions to create the container, by specifying the base image of the container and any subsequent changes needed to that specific image for the specific container [51][53].

The main modules installed in the container via pip or conda and its versions are specified below in Table 2-4.

Table 2-4. Most important modules used in this work and respective version.

Keras	2.2.4
TensorFlow	1.11.0
NumPy	1.15.4
NiBabel	2.4.1
xnatum	1.0.6

2.5.3 JUPYTER NOTEBOOK

To make this thesis possible and create the DL model to ease the researchers' work, an IDE was needed to provide a place where code could be written and tested. However, instead on a common IDE, Jupyter Notebook was used. Jupyter Notebook is an open-source and browser-based tool that supports workflows, code, data and visualization [54]. It can handle text, code and output of said code in the same file which can be then exported to be shared in an editable way or even in formats like pdf. The files created by Jupyter Notebooks are called notebooks [6]. The notebooks also allow long experiments to be split into smaller pieces that are executed independently giving the possibility of running only a part of the code that did not work instead of losing valuable time running all the code again [6].

Instead of the plain version of Jupyter Notebooks, JupyterLab was used in this thesis, the next-generation user interface for Jupyter Notebooks. It offers all the familiar building blocks of the classic Jupyter and some more in a dominant user interface [55].

2.5.4 KERAS AND TENSORFLOW

Keras is a high-level yet user-friendly library, that provides high-level building blocks for developing Deep Artificial Neural Networks [6]. It was used together with TensorFlow, due to its ease of use and extremely powerful structural bases for the network. Using Keras and TensorFlow allowed fast prototyping when trying to achieve the perfect network for the case study. Another advantage of these

APIs was the possibility of working with both CPU and GPU. As such, the GPU was used to process the large amount of data when creating, compiling and most importantly training the neural network model while, at the same time, doing it faster when compared to a CPU. The Figure 2-24 shows a representation of the software and hardware stack when using Keras and TensorFlow. As seen in the image, Keras is run on top of TensorFlow.

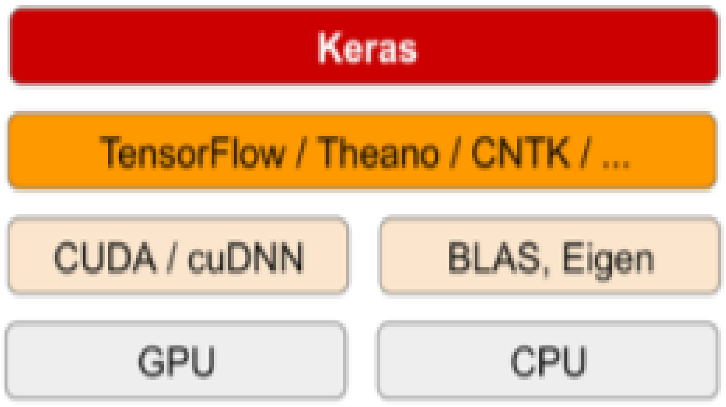


Figure 2-24. Software and hardware stack using Keras and TensorFlow [6].

2.5.5 NIBABEL

NiBabel[3] is a Python package that provides easy read and write access to some of the most common neuroimaging file formats such as DICOM. Among the accepted formats is also the NIfTI format which is the format of the MRI images of the dataset used. As such the library was used to access the image data to be used throughout the process. To do so, the "load" method of the NiBabel library was used. It takes the path of the image to load as an argument and returns an object with the information contained in that NIfTI file. The returned information consists of a header (which contains information about the MRI study) and the data of the study itself, i.e. the image. This data can then be extracted from the object using the "get_fdata" method. Doing so returns a NumPy array with the image data. This data was afterwards pre-processed before being used to train the model as will be discussed in a further section.

[3] At the moment of writing this dissertation, the most recent NiBabel version was 2.4.1 [62].

This library was also useful to save the prediction of the model also in NIfTI format. To do this, two methods were utilized, firstly a NIfTI object was created using the method “Nifti1Image” receiving, as one of its input arguments, the image data in the format of a NumPy array to be converted and saved to NIfTI. Finally, the “save” method takes the NIfTI object and saves it to the specified path.

3 Spatial Normalization of the Brain

This section describes the dataset used to train the Artificial Neural Networks used to learn how to perform the spatial normalization on the inputted brain MRI studies as well as the whole workflow that lead into the results. It also describes some of the attempts to recreate the existing tools in a Deep Artificial Neural Network approach before the final Deep Learning model. Finally, this section describes the results obtained in this work with this experiment.

3.1 SPATIAL NORMALIZATION

As explained in a previous chapter (2.2), when dealing with brain images, conclusions can't be taken by simply comparing the brains on each study directly. In order to do so, a spatial normalization must be applied which allows having all the exams in the same dimensional space. To normalize the brains a template is needed: it provides a target to compute all the images into. There are two well-known templates, the more broadly used MNI templates and the Talaraich Atlas. Furthermore, a tool is required to apply these templates to a given MRI study. Applying the MNI templates or the Atlas to the brain images normalizes the brains spatially and enables the possibility of taking conclusions regarding the areas of the brain of the different individuals in the brain dataset. However, the normalization has a disadvantage, it usually takes a lot of time. This is due to a very complex set of transformations that must occur for the normalization to be complete.
With that in mind, this section of the dissertation explores the attempts at trying to develop a new tool to perform the spatial normalization process faster by making use of Deep Artificial Neural Networks.

3.2 MATERIALS

As stated before, the goal of this work is to develop a DL model to normalize the MR image of the brain. As such, one of the most important materials used was, in fact, the MRI studies themselves. The dataset used, stored in a XNAT server, contained a total of 213 studies, properly anonymized to ensure the data of the patient stayed secure, in NIfTI format which were divided in three groups of 71 where each study in one group had a corresponding one in the other two. The groups were the following: a group of original studies (the data obtained in the MRI without spatial normalization), one of brain extracted studies (obtained from the corresponding original study) and finally one of Normalized studies (result of the normalization of the first group by an already existing tool). Of these groups, the 'original' was used as input for the Deep Learning model and the 'normalized' as an output. The brain extracted group was only used to pre-process the input for the model.

One of the studies contained in the dataset, more specifically one that has been normalized, can be visualized in Figure 3-1 where all slices of that study can be seen.

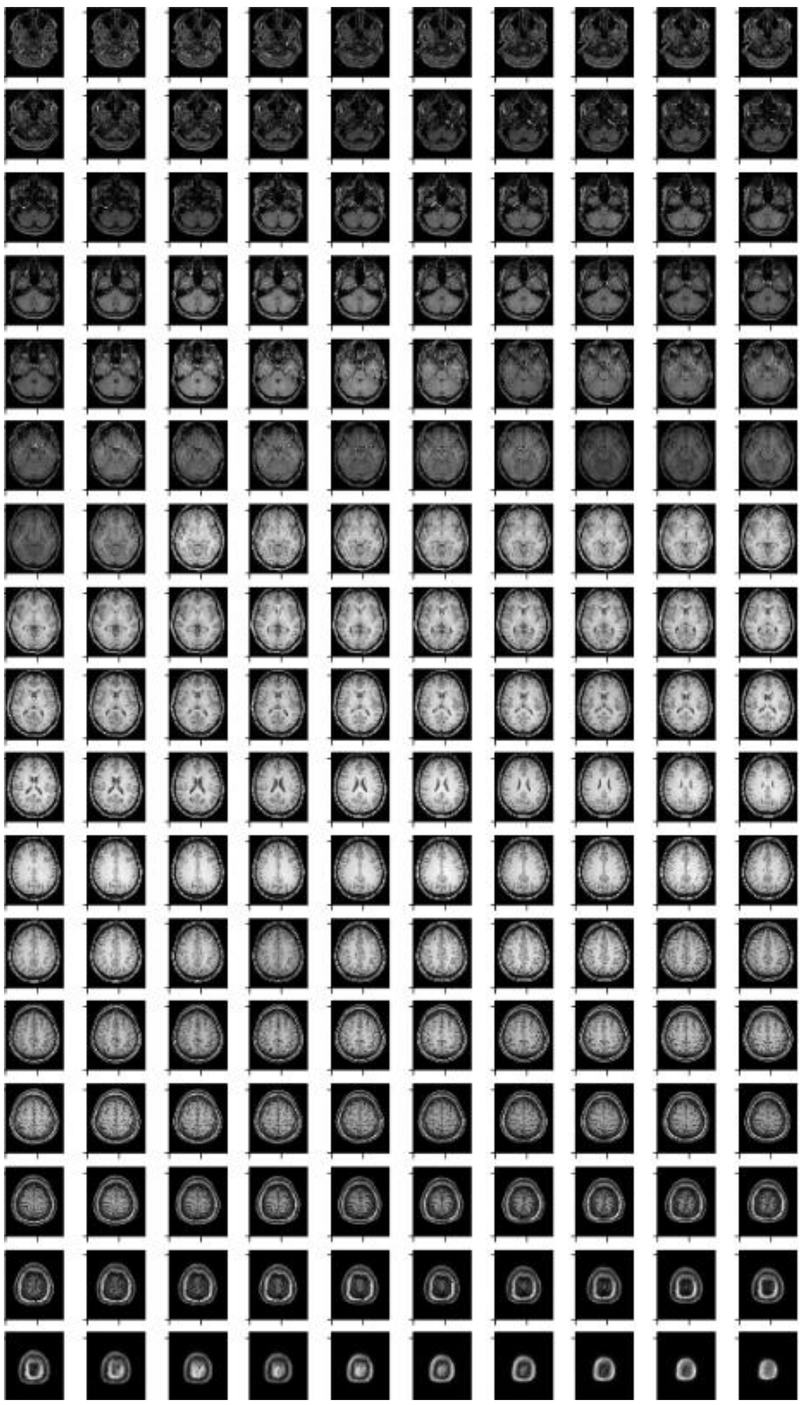


Figure 3-1. Visualization of all slices in one of the normalized studies.

To create and train said model, a programing language was needed as well as a suitable environment to work, Python was the chosen as the programing language, with very comprehensive imaging libraries as well as DL ones. Jupyter Notebooks was used as an environment in which the tools described in this dissertation were created.

The tools and materials described before are useless if there is no machine to use them and the capabilities of said machine may influence the results obtained. As such, the settings of the machine used while working to develop this dissertation can be seen in the Table 3-1.

Table 3-1. Specifications of the system used to develop the deep neural network models.

OS	Ubuntu 10.04 LTS (64 bit)
CPUs	Intel® Xeon® CPU E5-1650 V2 @ 3.5 GHz – 6 Cores
RAM	64 GB
Disk space	2 x 2TB disks 1 x 512GB disk
GPU	NVIDIA QUADRO P6000 GPU (24GB of GDDR5X dedicated memory) Cuda Parallel-Processing Cores 3840 FP32 Performance 12 TFLOPS

3.3 DATA PREPROCESSING

The first step to create the DL model was to prepare the dataset in order to be compatible with the model that was, at the time, yet to be developed. This preprocessing of the dataset ensures the data to be loaded in the model is tailored to suit the model like a glove. If the preprocess wasn't done, the data outputted by the model could not be as expected due to improper data in the input. By separating the preprocessing and loading into different stages we can also preprocess just once and load the already preprocessed dataset ready to use in a matter of seconds before the training of the model instead of having to spend time processing the data each time the model has to run.

3.3.1 PROCESSING

The preprocessing step is one of the most important, as this is the step that prepares the data to work perfectly with the model.

To achieve a perfect match between the data and the model, several steps are required. Firstly, a supported data format is required, the deep neural network model won't simply accept the MRI images as they are. To solve this issue, a python library called NiBabel which can handle files in NIfTI format, is used to extract the data from each exam into NumPy arrays. The next issue involves the data itself, the model needs to have meaningful data at its input, in other words, not any data will achieve good results. Therefore, the data from the exams, now in NumPy arrays, needs to be normalized to achieve good and repeatable results. This can be done by realizing several simple operations on the arrays. At the normalization step, the borders of the not normalized images are cut, leaving the brain intact. This insures the brain portion of the image is intact, yet it saves some space in memory, which is useful when training the model. The next step is to make sure all images have the same dimensions. To solve that, padding is added to the images as zeros to reach a final size. It was added to the images instead of removing to maintain as much valuable information as possible, not tampering possible crucial values. Finally, the values of all the arrays must be between 0 and 1, insuring that the model can take conclusions from all the images independently of their maximum or minimum value. As the minimum value of all the arrays was already 0, the maximum value was the target for the normalization. The solutions to present values between the required interval was to divide each not normalized image by its maximum and the corresponding normalized by the same value. That makes sure the maximum value is 1 and enables repeatability.

At last, the now preprocessed dataset, is separated in train and test NumPy arrays to be saved in four NumPy files (X_train.npy, Y_train.npy, X_test.npy and Y_test.npy) to get a faster data loading when needed. By doing this the data can be preprocessed only once and loaded much faster anytime the model needs to train. To separate the data in training and testing sets, 50 of the studies were used for training (about 70% of the dataset) and the remaining 21 (remaining 30%) were used for testing. The train set is used to train the model and the test set will evaluate the model capabilities whilst training.

The Figure 3-2 represents, in general, the operations made by the function for preprocessing the data and then splitting it into the files.

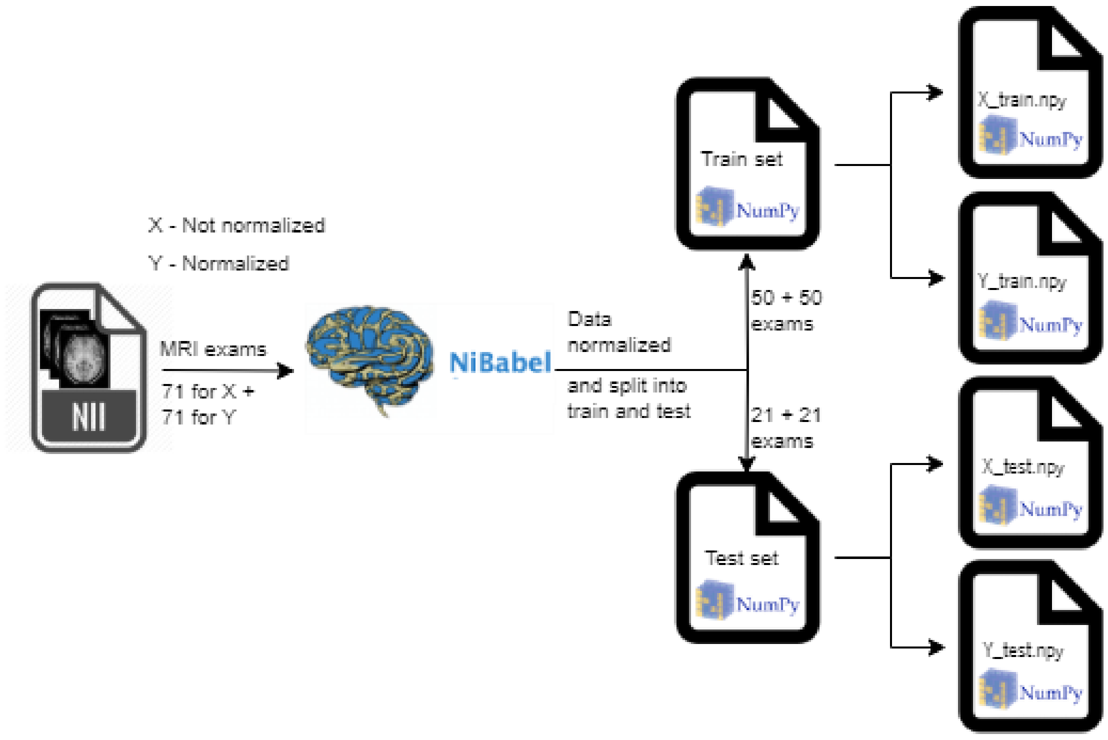


Figure 3-2. General process of the data preprocessing.

3.3.2 LOADING

To load the pre-processed data into the model, a function was created to ease the process. That function's job, as represented in the Figure 3-3, was to access the four NumPy files mentioned earlier, created when preprocessing the data using the function described in the section 3.3.1. It then returns the data as four different NumPy arrays, ready to be used in the DL model. The arrays are named: X_train, Y_train, X_test and Y_test. The first two are, as the name implies, for the training of the model and the last ones for both testing and validation.

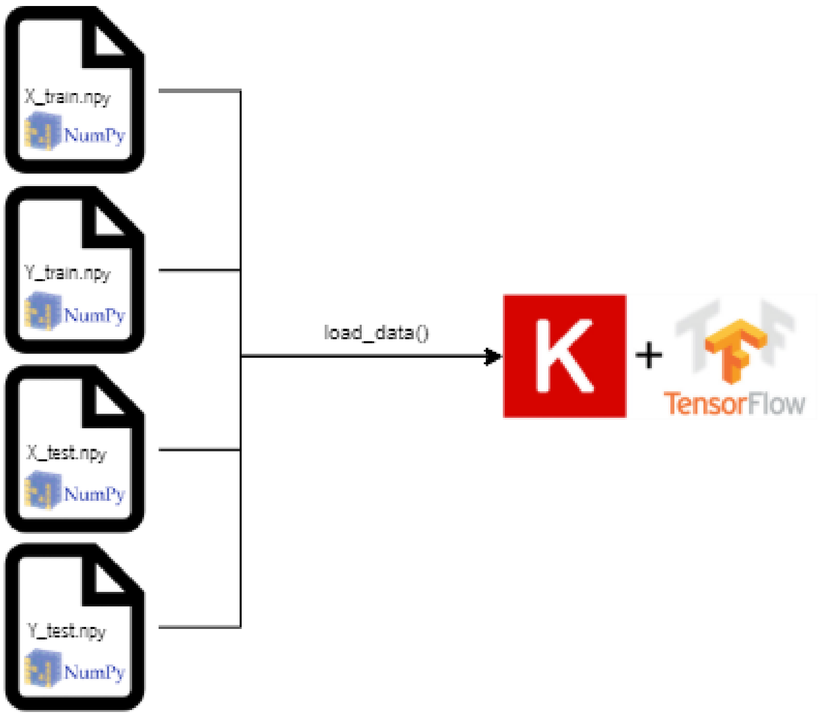


Figure 3-3. Data loading process diagram.

After successfully loading the data from the files, the next step was to create and compile the models to train them afterwards. The creation and training of the models for each experiment made was slightly different due to the requirements of each model tried. As such that information is split between the two experiments, using in one a U-Net based architecture and a GAN in the other.

3.4 BRAIN SPATIAL NORMALIZATION USING A U-NET BASED ARCHITECTURE

This first experiment used a U-Net based architecture for the Deep Artificial Neural Network. This network was created and tested as described in the following sub-chapters. Although this experiment didn't solve the initial problem of developing a substitute for existing spatial normalization tools as it didn't achieve the best results it did, however, get the shape of the brains correct as can be seen further below while evaluating the model in 3.4.2.

3.4.1 CREATING AND TRAINING THE MODEL

The Deep Artificial Neural Network was created in a separate python module named "modelo_unet.py".

To try solving the spatial normalization problem, at first an Autoencoder architecture was used, however, this structure was quickly changed to a U-Net based one due to its more powerful capabilities.

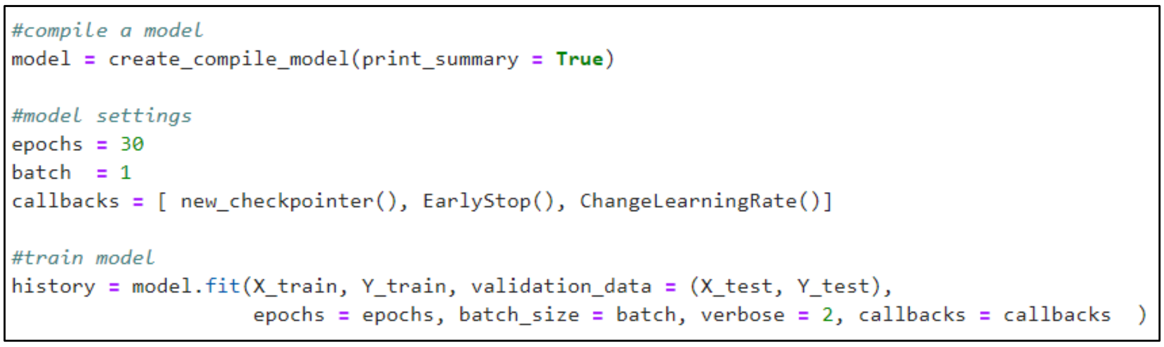
For creating and compiling the model contained in the first module we simply call the "create_compile_model()" function contained in said module. This function starts by adding layers to create the U-Net based structure of the model. It then compiles the model using Dice Similarity Coefficient (DSC) as loss function and Adam (Adaptive Moment Estimation) with a lr of 1e-6 as an optimizer. As activation functions, ReLU and Leaky ReLU were used strategically. Several learning rates were experimented with but the specified above was the one which obtained the best results.

This module also contains helpful parameters to be used such as the input size as well as callbacks to use later while training the model. The callbacks used in the first experiment were:

- ModelCheckpoint – saves the weights of the model at defined points of the training, in this case saves the best possible weights of the training;
- EarlyStopping – this callback monitors the model training and automatically ends training if the model is not learning as it should;
- ReduceLROnPlateau – Adapts the learning rate (lr) of the model to adapt its learning and hopefully achieve better results;
- PlotLossesKeras – Implements a plot which updates each epoch to show the model history, i.e. accuracy and loss at each epoch.

All these callbacks are very helpful and improve the chances of achieving good results with the model.

In the Figure 3-4 the code to create and train the model can be analyzed. First the model is created and compiled using the function explained before. Next, some important settings are specified such as the number of epochs to train the model as well as the batch and the callbacks to use (if needed). As said before, the functions present in the callbacks part of the settings are specified in the model module.


```
#compile a model
model = create_compile_model(print_summary = True)

#model settings
epochs = 30
batch  = 1
callbacks = [ new_checkpointer(), EarlyStop(), ChangeLearningRate()]

#train model
history = model.fit(X_train, Y_train, validation_data = (X_test, Y_test),
                    epochs = epochs, batch_size = batch, verbose = 2, callbacks = callbacks  )
```

Figure 3-4. Steps to create and train the U-Net based model in the code.

Both the number of epochs and batch size were changed to influence the training of the model hoping to improve the model performance. Where the number of epochs represents the number of passes through the training samples whilst training. The batch size essentially represents the number of samples presented to the model at each time. As expected, a higher batch size represents a higher number of images being presented to the model and as such more memory space is needed to accommodate all the data.

Even though the model was trained using a high-performance GPU the training process took much time due to the size of the dataset, in the case of this architecture, about 6 hours with the specified settings.

After training using the GPU, the model is evaluated to understand if it performed as expected. It's structure in ". json" format and weights in ".h5" format using h5py package[4] are then saved in disk to further reference. Thanks to one of the callbacks used, the weights of the model at the best stage were also saved. At this stage, the plots of the loss and accuracy of the model are shown or, thanks to the PlotLossesKeras callback, these are shown throughout the training process in constantly updated plots. These are then used to measure the performance of the model.

3.4.2 MODEL EVALUATION

After training with the given dataset, the model was evaluated to understand its performance. This was done using both the evaluate and predict model. The evaluate model evaluates the model and outputs its scores, loss and accuracy, after the training. The predict method uses the model and predicts an output to the given input. The output predicted by this model using this method, to a

[4] H5py is a Pythonic interface to the HDF5 (Hierarchical Data Format) binary data format

random input can be seen in the Figure 3-5. In the figure the input is the column ‘FSL in’, the output by the FSL is ‘FSL out’ and the model output is ‘DL out’.

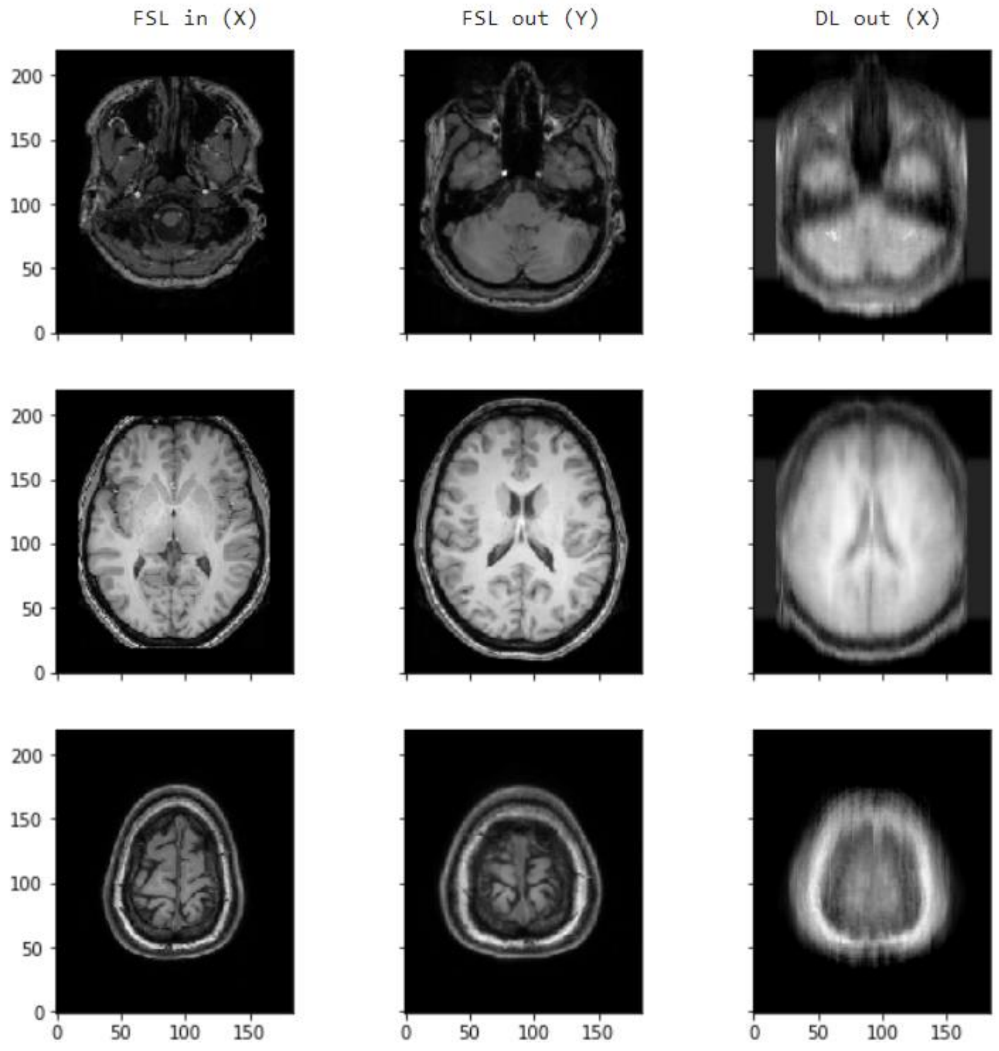


Figure 3-5. Comparison between the original image, image obtain trough FSL and prediction of the trained model.

To achieve a perfect result, the third column of the image should coincide with the second as the second is the output of the normalization obtained with FSL and the third represents the output of the trained DL model. As can be seen, the results are not perfect but are close to the target image as the shape of the output coincides almost perfectly to the expected. Especially when inspecting the top row, we can see the model predicted the features correctly. Therefore, it achieved good results but not good enough to compete with the already existing tools.

By analyzing the dice loss function of the model, which as explained before measures the overlapping areas of the images, its value gets very close to 0, meaning the model performed good while deforming the original brain to achieve the final form even though the image in a whole does not look as expected. The graph of the loss function throughout the training of the model can be seen on Figure 3-6. The accuracy graph however, will not be shown as it's not a good metric for how good the model learned in this specific case. This is because even if two corresponding pixels have very similar values but not the same, this is considered a wrong prediction even though it would be acceptable. That being said, the accuracy achieved by the model was very low and constant despite the fact that the model got the shape right as seen above in the Figure 3-5.

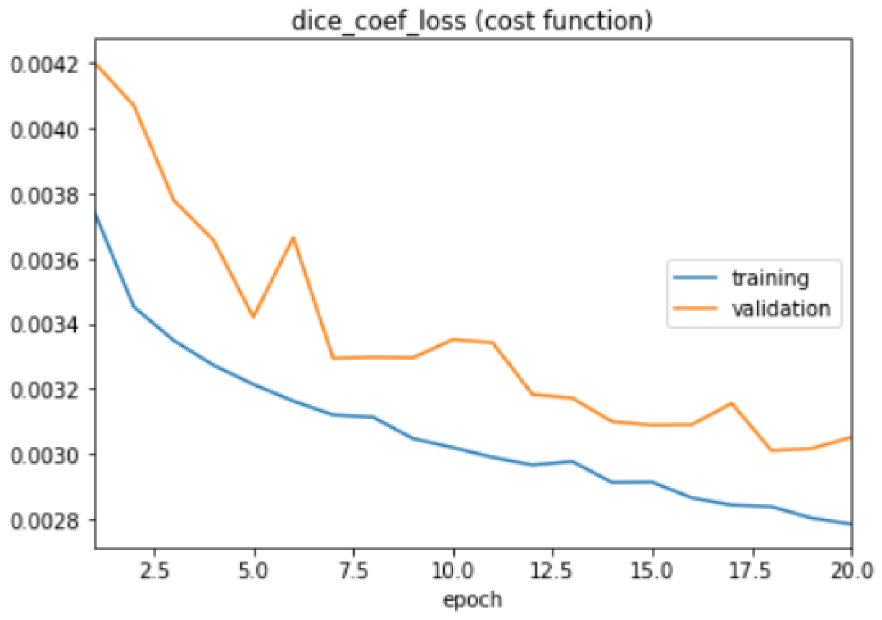


Figure 3-6. Evolution of the loss when training the model.

The specific architecture used in this first model can be seen in a representation the Appendix B – U-Net based architecture used.

Since this experiment didn't solve the problem, despite getting the shapes correctly, a new approach was made. This new approach, using a GAN architecture would try to solve the problems the first architecture had and hopefully achieve better results.

3.5 BRAIN SPATIAL NORMALIZATION USING A GAN ARCHITECTURE

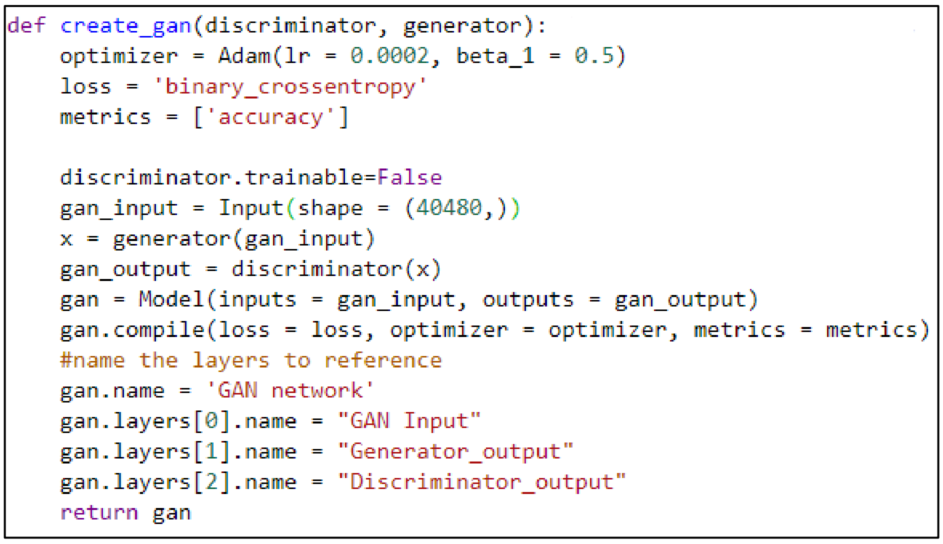
As stated before, to attempt to improve the results obtained in the previous experiment (see section 3.4), this experiment followed a new approach using a GAN architecture. This chapter describes the creation and testing of said network and its evaluation, showing some of the results obtained.

3.5.1 CREATING AND TRAINING THE MODEL

Although having some similarities with the first experiment's network, this type of architecture is more complex yet simpler at the same time. As all GANs, the main GAN network was created by 2 smaller and independent ones, the Generator and Discriminator. These were then combined to create the so-called GAN, the main network. As explained in more detail in the section 2.4.2.3, the Generator takes noise as input and tries to reproduce an image which, if the model is tuned, should be close to the real ones. This essentially means the GAN can generate an expected image from that noise. To create the generator, a function called "create_generator()" is used. It creates the sequential model of the generator by adding its layers in sequence and then compiles it. To compile the generator, Adam was used as optimizer and binary crossentropy as a loss function. A lr of 0.0002 was established in the optimizer. Regarding the Discriminator, it was created the same way, a "create_discriminator()" function was used which adds layers to create the discriminator. The discriminator was compiled with the same parameters as the generator, Adam with an established lr of 0.0002 as an optimizer and binary crossentropy as a loss function.

Both the Generator and Discriminator used Leaky ReLU as an activation function with 0.2 as parameter except for the last layer, where the sigmoid function was used in order to maintain the final output between 0 and 1.

Finally, the GAN itself is created by using the two models built as described before. As seen in the Figure 3-7, the GAN is created using the "create_gan()" function. Its input is predicted by the Generator and then passed to the Discriminator which tries to guess if the image it received was, or not, a real image. The model is compiled using the same settings as the two smaller ones and the layers are named to better understand the model.



```python
def create_gan(discriminator, generator):
    optimizer = Adam(lr = 0.0002, beta_1 = 0.5)
    loss = 'binary_crossentropy'
    metrics = ['accuracy']

    discriminator.trainable=False
    gan_input = Input(shape = (40480,))
    x = generator(gan_input)
    gan_output = discriminator(x)
    gan = Model(inputs = gan_input, outputs = gan_output)
    gan.compile(loss = loss, optimizer = optimizer, metrics = metrics)
    #name the layers to reference
    gan.name = 'GAN network'
    gan.layers[0].name = "GAN Input"
    gan.layers[1].name = "Generator_output"
    gan.layers[2].name = "Discriminator_output"
    return gan
```

Figure 3-7. Code for creating and compiling the GAN model.

The step of creating the model is, despite having to create several models for the main model to work, very similar to the model in the first experiment. The training process however, is completely different. In the case of the first experiment's model, the training used only one step: applying the fit method to the model. This time, the training is much more complex involving several steps as the training, in this case, is alternating between training only the Generator and training only the Discriminator.

To perform the training, we start by loading the data. A small change to the arrays is made, the data in them is reshaped to, instead of having several 2D images in the arrays, having several one-dimensional arrays. Note that the information is still present and can be rearranged again to become 2D images. As the images had dimensions of 220x184, they were resized to a vector with a length of 40480 (220*184). After this slight modification, the 3 models are initialized, first the Generator and Discriminator and then the GAN which uses the first two to be created. The training process then begins, using a for loop to repeat the instructions for each of the defined epochs. Firstly, random images from the X_train array, now in the 1D vector form, are selected i.e. a random amount of non-normalized images. The number of images selected is represented by the batch size. These images are then inputted to the Generator as 'noise' and the output is predicted to get the generated images. The same number of images is then selected randomly from the Y_train array, the normalized images, to represent 'real' images. The generated images and real images are then concatenated in a single array and a matching array, which contains labels identifying the images as real or generated, is created. Now, with a set of images and corresponding labels, we train the Discriminator, after setting it as trainable by indicating Discriminator.trainable as True. This is done using the method train_on_batch. This method takes two arguments, the images and the respecting labels, and then trains, for this epoch, the Discriminator. As the Discriminator finished its training, for the time being, it is set to not trainable to maintain its weights in the next step. Now, a new set of random non normalized images is chosen as well as the respective labels, this time to train the GAN and, consequently the Generator as the Discriminator is set to non-trainable. However, there is a twist this time, the labels created for the chosen random non normalized images are set as these were normalized ones. By doing so, we try to trick the discriminator into thinking they are indeed the correct ones. Once again, the train_on_batch method is used, this time on the GAN, using the mentioned images and respective labels. If all goes as expected, the Generator will, at some point learn how to create the normalized images using the given 'noise' (non-normalized images) in a way that the Discriminator can't recognize they are not the real images, meaning the Generator of the GAN should learn to generate the expected images in a convincing way.

This process is then repeated a set number of times, determined by the number of epochs established for the model train.

Just like in the first experiment, both batch size and epochs were changed to experiment with the model and analyze the changes made to it. Changing the batch size and epoch number influences greatly the time needed to train, taking for example a few hours to train 1000 epochs or even several days to train 20000 epochs.

After finishing the training using the GPU, the model had to be evaluated to understand if it learned correctly.

3.5.2 MODEL EVALUATION

To help evaluate the GAN a function was created to predict, using the Generator of the GAN, a set quantity of samples each set number of epochs to keep track of the GAN's performance. The function in question, "plot_generated_images()" takes as parameters, the epoch at which it was called to name the output plot of predicted images, the Generator to predict them from the test set and the test set itself. The number of images to be plotted can also be passed to the function using the "examples" parameter, which by default is set to 100.

As an example, Figure 3-8 shows side by side, the output of the function, while training the model, where the plot A was after the first epoch and B after 1000. To better fit the pages of this dissertation, not every plotted test image is shown in the figure, showing a matrix of plotted images of 5x5 instead of the 10x10 generated by the function.

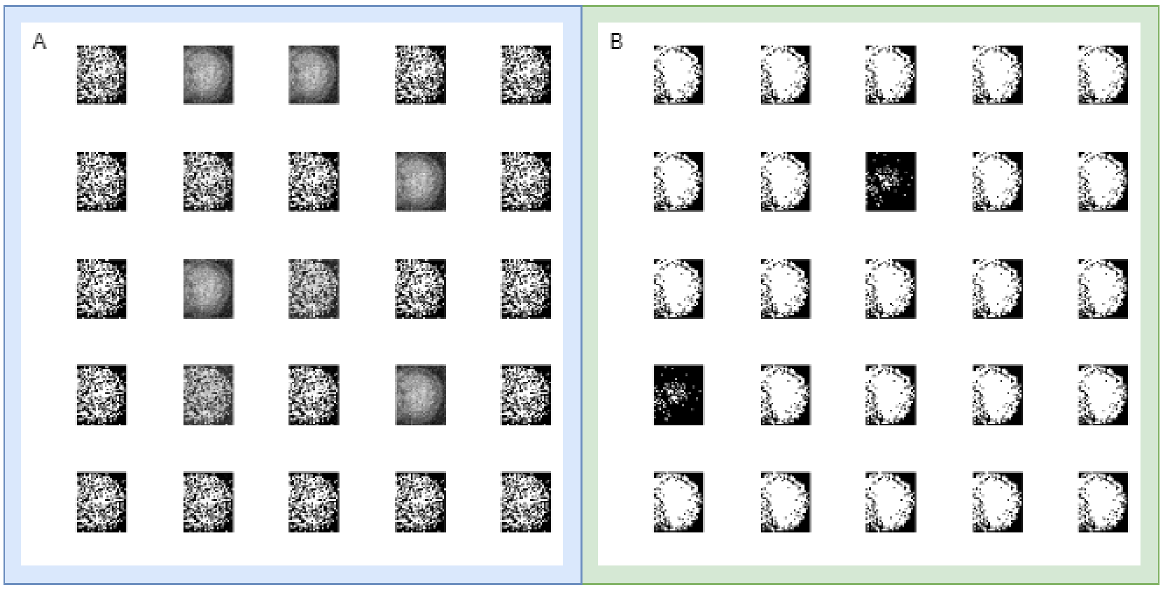


Figure 3-8. GAN output sample after first epoch (A) and 1000th epoch (B).

The GAN was also tested in its final epoch to observe the result of the training. After training, the model was again used to predict a random study of the dataset and was compared with the respective original study and exam outputted by the FSL, i.e. the reference output, to understand if the model could perform the spatial normalization of the brain as was expected. One of the results of this final test can be seen below in the Figure 3-9.

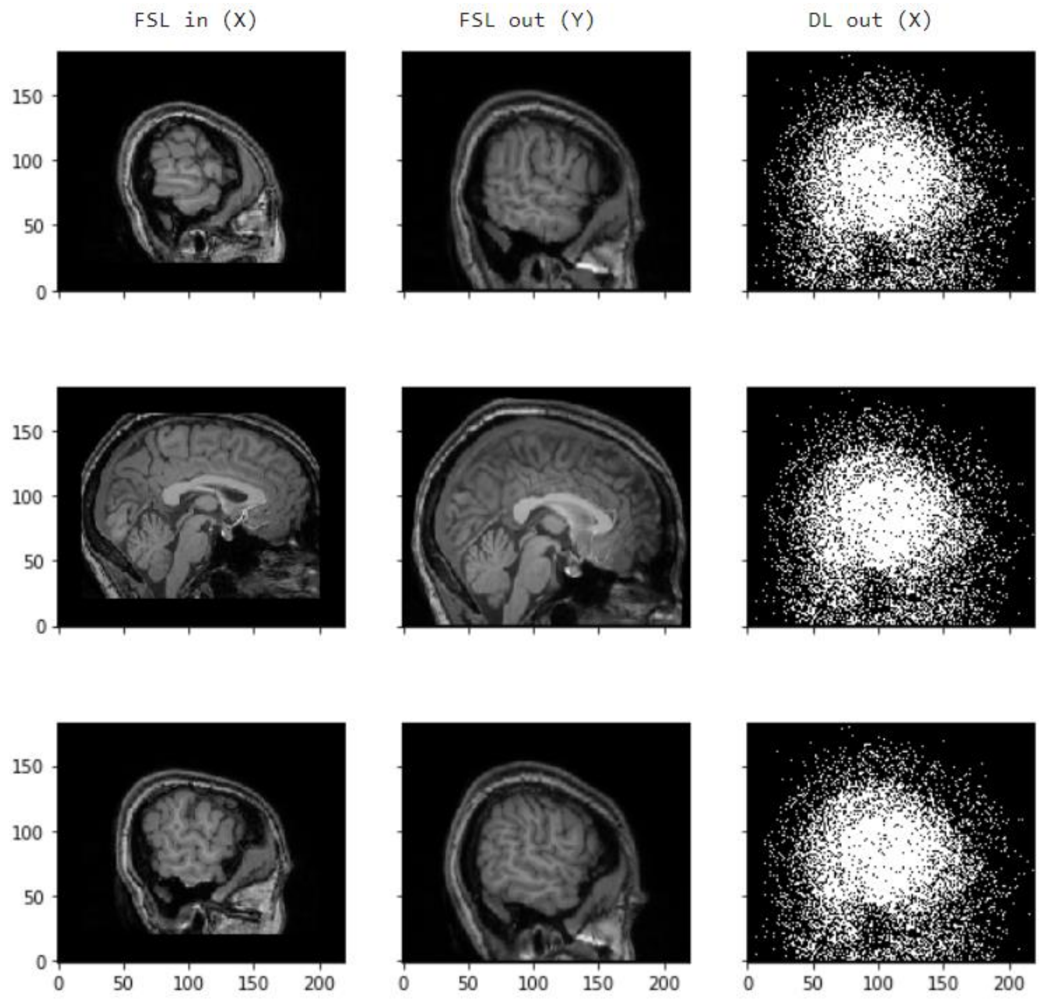


Figure 3-9. Comparison between original image, expected output and GAN output.

The figure shows, the model used to predict that study was not successful at achieving a good normalization but once again, just like the previous experience with the U-Net based architecture, the shape was approximated by the model. The result also shows that, in comparison, the U-Net based architecture had a better performance and learned more features than the GAN as it achieved better results.

3.6 RESULTS AND DISCUSSION

Several parameters can be used to evaluate a model's characteristics and performance when predicting its output. This evaluation can be as simple as comparing the class predicted by the model to what the true class should be and, if both are equal, the model performed as expected, or it could be something more intricate such as comparing the areas of two features to see if they are as close as possible. Different models call for different approaches so, different metrics were used to correctly evaluate the models' performance. Usually the main metric to evaluate the model is the accuracy metric which indicates the percentage of correctly predicted outputs.

However, despite seeming the best metric to evaluate a model, accuracy is not always the best choice. In the case of spatial normalization for instance, depicted in this dissertation, this metric doesn't stand for a valid method to evaluate the model. This is because in this case accuracy does not understand the values it is dealing with. It analyses, in this case, each pixel of the image in the output, if the value is not the same for that pixel, when comparing its expected value with the predicted one, the accuracy metric doesn't understand the context of the value, it just outputs if the value was correct or not regardless the fact that close values would be accepted as a good prediction. As an example, Figure 3-10 shows two images side by side, where one of them has all its pixels tampered from the original as a unit was added to each of the pixels. As a reference, the image on the left is the original and the one on the right is the tampered version. Even though the images are considered different to the computer and, as such, are considered as a bad prediction by the model, they are indeed the same as they are intended to be interpreted by a human and not a computer.

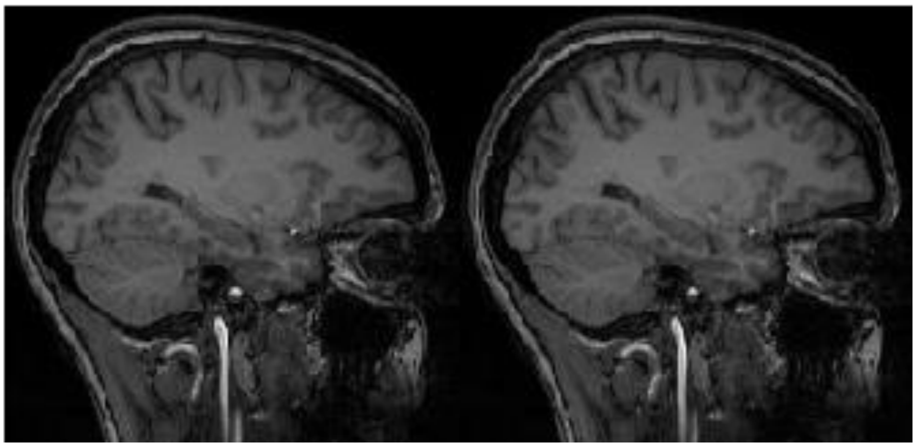


Figure 3-10. Comparison between two images with one unit added to all pixels.

Naturally, if the values are much different from what they should, the image is indeed wrongly predicted but a range of values close to the expected could be accepted as a good prediction. Also, an accuracy of, for instance, 90% or higher would probably mean the model is working as expected as it does indeed usually get the output predicted correctly to reach such high accuracy. However, low accuracy doesn't necessarily mean the model is not predicting correctly as already shown.
To overcome the confusion generated by the accuracy metric a new way to evaluate the results was necessary. To correctly evaluate the model performance the dice loss was used in the U-Net based model. This loss function is based on the Dice Similarity Coefficient (DSC) [44]. The Sørensen–Dice coefficient or Dice Similarity Coefficient is used to measure the overlap of two areas. If the overlap is perfect, the result of DSC will be 1 meaning 100% is overlapped. On the other hand, a DSC of 0 would mean areas totally spaced apart or a 0% overlap. Figure 3-11 illustrates the DSC depending on the overlap percentage.

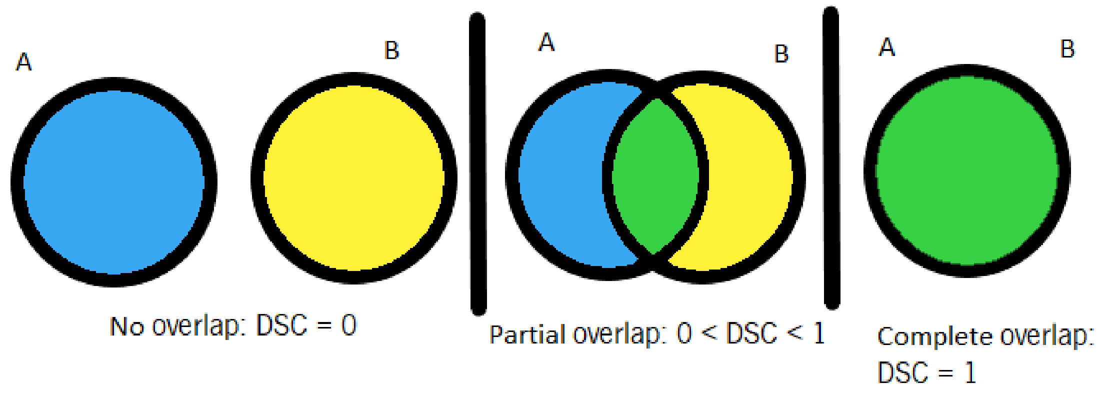


Figure 3-11. Dice Similarity Coefficient (DSC) for different overlaps.

The Dice Loss, opposite to the DSC it is based on, tends to 0 as the overlap improves. A Dice Loss of 0 would mean a perfect overlap and consequently a model predicting correctly its output. The Dice Loss value obtained with the U-Net based approach was 0.00313, meaning the overlap was almost perfect whereas the accuracy did not evaluate the model as it should, as it was a constant of 16.69% accuracy throughout the training of the model without any alterations. The side-by-side comparison between the expected result and the obtained result by the U-Net is shown in Figure 3-12.

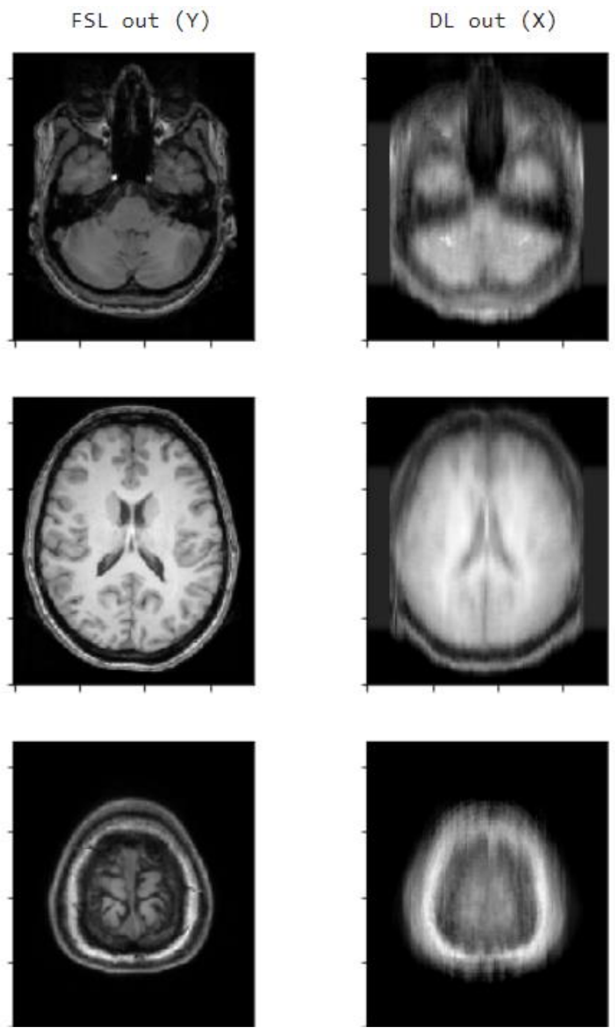


Figure 3-12. Comparison between the expected (Left) and the obtained result by the U-Net (Right).

For the GAN approach an evaluation of the results was harder to do as several models have to be trained to train the main GAN network.

In the case of the GAN approach, the results were analyzed simply by comparing the output of the network with the expected one. As seen in Figure 3-13, the results are promising like the ones obtained in the U-Net based architecture where the shape is somewhat close in general. However, this time, the output shows that maybe this type of architecture isn't up to the task of handling the spatial normalization.

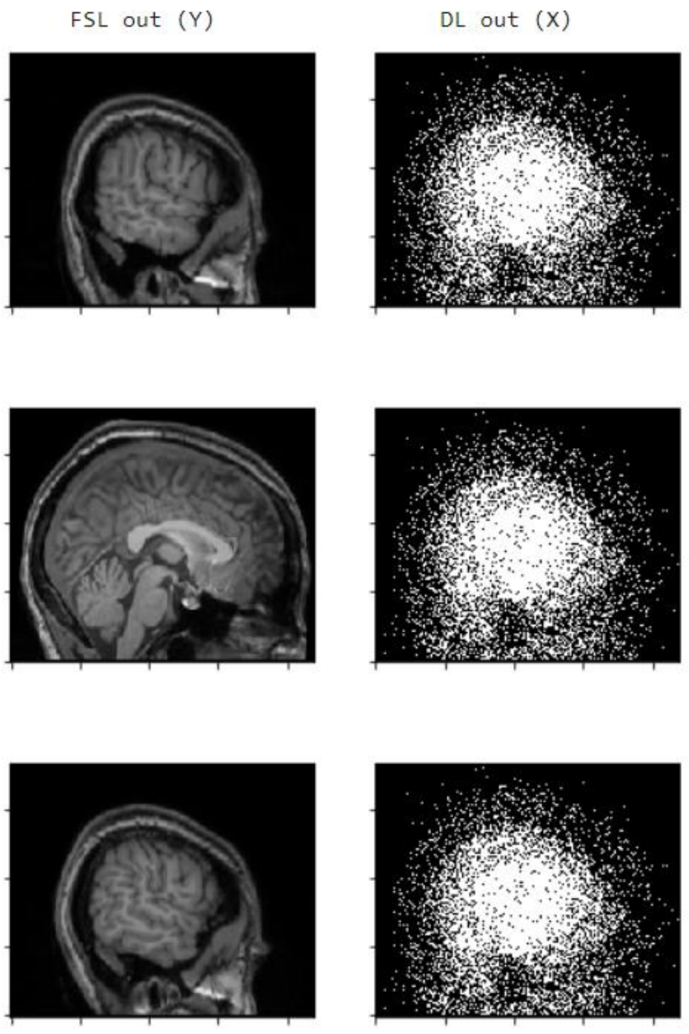


Figure 3-13. Comparison between the expected (Left) and the obtained result by the GAN (Right).

Changing the architecture of the GAN could impact its performance in a way that could achieve better results but, with the experiments made, the U-Net based architecture seems to achieve far better results and, with some more alterations could possibly achieve a proper brain spatial normalization.

4 Brain Atrophy Classification

This section describes the dataset used and models prepared to predict the atrophy stage of brain MRI studies. It starts by introducing brain analysis and then describes the materials used (dataset and system). Next it describes the approach made with the model described in detail, explaining how it was created, trained and, most importantly, how it was evaluated to understand its performance. Finally, the results are discussed for a better understanding of what was obtained.

4.1 BRAIN ANALYSIS

Distinguishing between the different neurodegenerative causes of dementia is vitally important to allow affected individuals and their families to access appropriate treatment, support and care [56]. Structural neuroimaging is widely available and recommended as part of the clinical evaluation in all patients with suspected dementia, MRI in particular allows for global and regional cerebral atrophy to be assessed [57]. A wide number of sophisticated methods of analysis is nowadays available to quantify global and regional atrophy from MRI [58][59]. However, almost no progress has been made to integrate these methods into clinical work streams. This is due to several reasons, but mainly special hardware requirements and long processing times [57]. Therefore, the main method for extracting useful clinical information in MRI studies for a diagnostic continues to be a visual scan assessment. However, without proper operational guidelines to identify the atrophy patterns with diagnostic value in dementia, a lot of potentially relevant information may be under-utilized [57].
A solution that could be integrated in the clinical workflow that would solve this would therefore be much helpful than to visually inspect every single scan to identify brain atrophy. This would, if properly implemented, save time both to the professionals as well as to the patients, where the patients could be diagnosed faster, and the professionals would have much more time to act accordingly.

4.2 MATERIALS

The goal of this experiment was to develop a DL model to classify, when presented with a brain MRI study, it's stage of atrophy from class 0 to 4 as the Medial Temporal lobe Atrophy (MTA) score (Figure 4-1). Class 0 would mean no atrophy detected in the brain presented and, on the other hand, class

4 would mean the worst stage of atrophy. According to the MTA score the classification is made as follows [60]:

- Class 0 - no Cerebrospinal fluid (CSF) is visible around the hippocampus;
- Class 1 - choroid fissure is slightly widened;
- Class 2 - moderate widening of the choroid fissure, mild enlargement of the temporal horn and mild loss of hippocampal height;
- Class 3 - marked widening of the choroid fissure, moderate enlargement of the temporal horn, and moderate loss of hippocampal height;
- Class 4 - marked widening of the choroid fissure, marked enlargement of the temporal horn, and the hippocampus is markedly atrophied, and internal structure is lost.

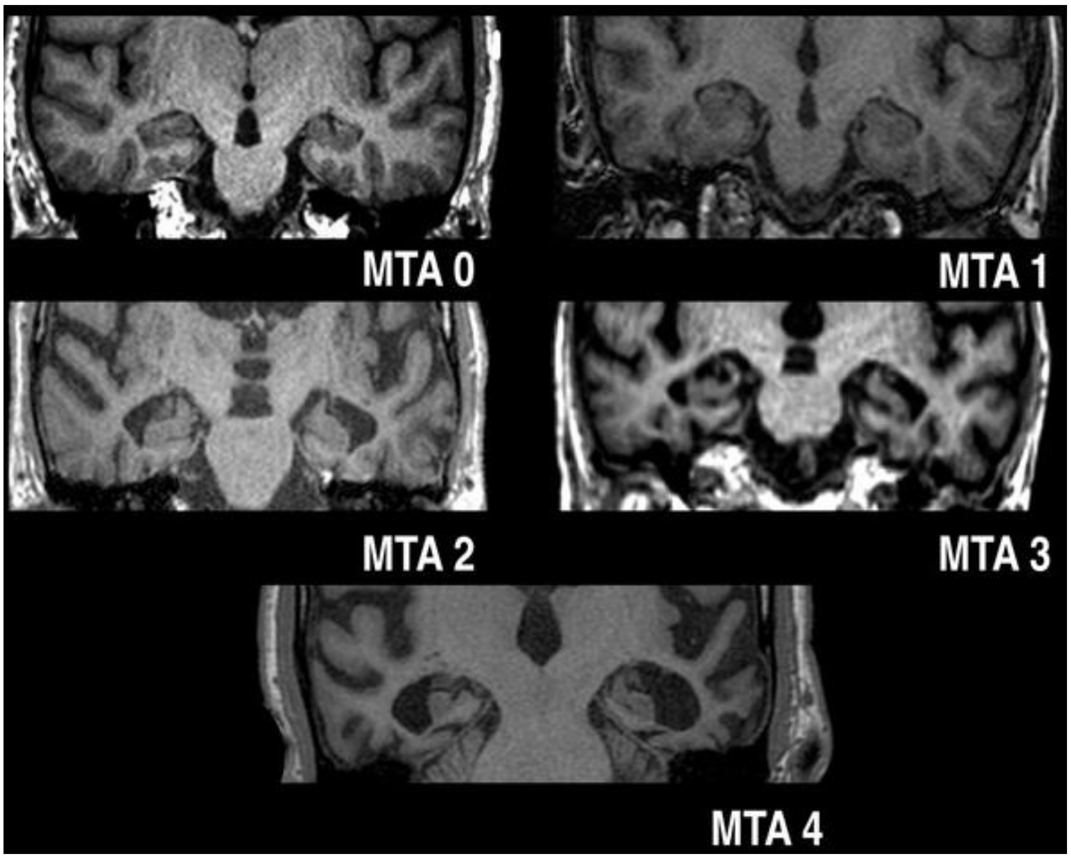


Figure 4-1. Visualization of images with different MTA scores [60].

The dataset provided, in T2 contrast, once again stored in a XNAT server, was properly anonymized to ensure the data of the patients was protected. Not all studies were used to this experiment as studies in different planes were available for each individual patient. As such, from the dataset provided, only the coronal plane studies, 129 in total, were used due to its abundance and uniformity among all of them. Each study was labeled by a set of two labels, corresponding to the average MTA score of the left side of the brain and the average MTA score for the right side of the brain when classified by 2 different experts.

Just like in the section 3.2, the dataset is useless if there is no machine in which to create the models and train them. As such, the settings of the machine used while working to develop this experiment can be seen in the Table 4-1.

Table 4-1. Specifications of the system used to develop the deep neural network models.

OS	Ubuntu 10.04 LTS (64 bit)
CPUs	Intel® Xeon® CPU E5-1650 V2 @ 3.5 GHz – 6 Cores
RAM	64 GB
Disk space	2 x 2TB disks 1 x 512GB disk
GPU	NVIDIA QUADRO P6000 GPU (24GB of GDDR5X dedicated memory) Cuda Parallel-Processing Cores 3840 FP32 Performance 12 TFLOPS

4.3 METHODS

4.3.1 DATA PREPROCESSING

When dealing with Artificial Neural Networks, Deep Artificial Neural Networks in this case, one of the most important steps is to do preprocessing of the data. This prepares the data to work perfectly with the network but most importantly insures consistency if the results are to be reproduced in some way. To preprocess the data to work with the Artificial Network several steps were required. The first step was to ‘exclude’ the unnecessary studies for this experiment, this means all studies that wouldn’t be used were ignored. Next, as 5 different classes exist, a for cycle was created to cycle all of them. In each cycle, the studies to be used were classified 1 or 0, as they had, or not the class being processed at that moment in the cycle. For example, a study labeled as class 2 atrophy in the dataset would represent a 1 for the class 2 model and 0 for the remaining.

Afterwards the studies needed to be normalized and, to do so, the max value of the studies voxels were used, after cutting the image, with a margin, around the brain, the images were divided by their max value. This made them all smaller, due to the removed part of the images and left them with values between 0 and 1 (due to being divided by the max value). At this point, several experiments were made to analyze how the results of the model would be affected. In some experiments the images were resized by adding 0s as padding, others left this to the Keras ImageDataGenerator to handle and finally, the use of the brain extracted images, i.e. the studies with everything but the brain tissue removed, instead of the studies with a margin around the brain removed, was tested.
After the normalization process the data was split to form the train and test sets. A percentage of 80% was used for the splitting, 80% for training and the remaining 20% for testing the model.
The Table 4-2 shows how the studies were distributed between the classes as well as the split between the train and test sets.

Table 4-2. Distribution of the data cases among the train and test of the different classes.

Class 0	Train	Test
True	8	1
False	95	25

Class 1	Train	Test
True	43	10
False	60	16

Class 2	Train	Test
True	45	15
False	58	11

Class 3	Train	Test
True	25	8
False	78	18

Class 4	Train	Test
True	11	3
False	92	23

Finally, the images resulting of this preprocessing needed to be saved. To do this, several folders were created to distribute the files. The Figure 4-2 represents generally the preprocessing step and shows how the figures were distributed among these folders. Each class had their folder, inside each of them, exists a train and a test folder. Each of the two has 2 sub folders, named 0 and 1. The images were distributed by the corresponding folders. For example, an image on the train set of class 2 classified as 1 would be saved in Class2 > train > 1 > example_image.png.

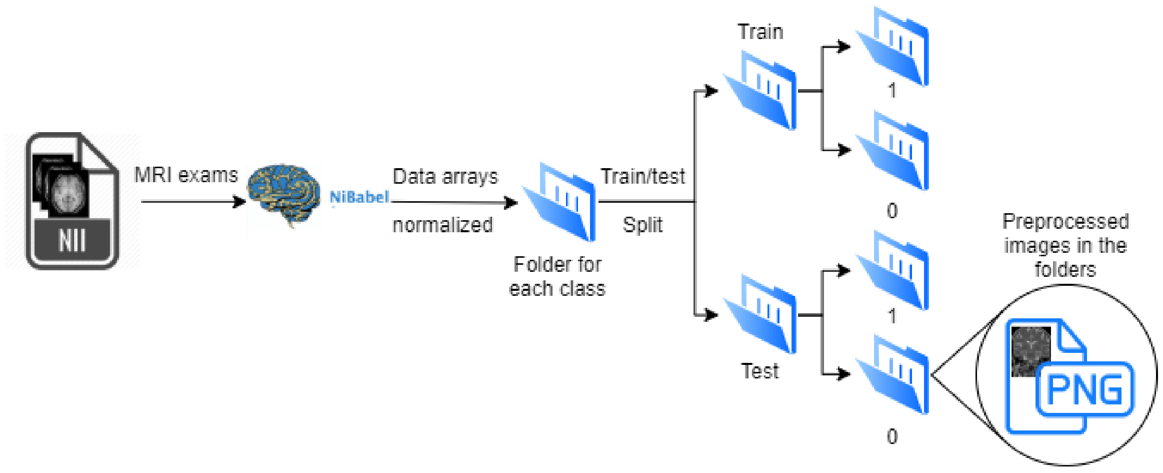


Figure 4-2. Preprocess diagram for the Brain Atrophy experiment.

4.3.2 CREATING AND TRAINING THE MODEL

This experiment started using an InceptionV3 based model. This was, however, quickly changed to a model based on the VGG19 architecture due to some problems encountered when adapting the first architecture. These problems were caused by one of the batch normalization layers of the InceptionV3 model initially used as a basis.

A model was created and trained for each of the 5 stages of atrophy and adapted to classify the corresponding stage in 2 classes as True or False, if it was present in the inputted image, instead of the 1000 classes of the original VGG model, more specifically to classify the brain atrophy stage. The general structure of the models used to classify the brain images regarding the stage of atrophy can be seen in Figure 4-3. As the classes were unbalanced and each MRI study had 2 labels regarding the left and right side of the brain, the model had to be tailored to each class. As such, the model had a custom block for each of the classes.

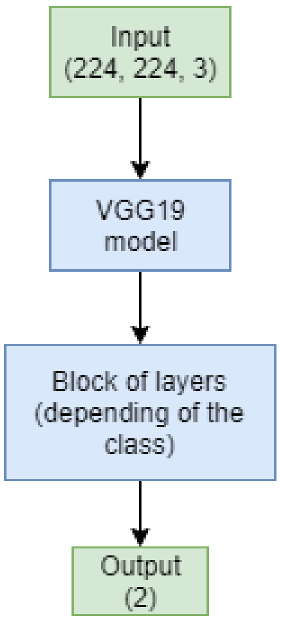


Figure 4-3. General structure of the model used for Brain Atrophy classification.

The models were all placed in a separate module named "modelo_atrophy_5class_complex.py". This module contained every function necessary to use the model, the most important is "create_compile_model()" which creates the model and compiles it using the settings specified in said module. This functions also takes one input parameter to differentiate the output model depending of the atrophy class to be predicted. It does so by calling other function that adds the class specific block to the model. The settings specified in the module to compile the model are the loss function to use and the optimizer. As an optimizer, SGD was used with a learning rate of 0.001 and for loss, categorical cross entropy was used. ReLU was used as activation function except for the last layer where SoftMax was used. This allowed the output to quantify the probability, according to the model, for the class outputted to correspond to the image inputted. To help train the model some more functions were placed in the module, specifically functions to create the callbacks to be used. The callbacks implemented were the same as described in the section 3.4.1.

Regarding the training of the model, it required some special methods to work, as the dataset involved a large amount of data, the Figure 4-4 shows the loading process for the training and testing steps. For that 'heavy lifting' loading, firstly, two ImageDataGenerator were created. It is a special Keras function that will load the data from a specified directory in a way that doesn't overload the memory space. That specific method is called flow_from_directory and, in that method some settings are specified to allow it to perform correctly. These settings include the size of the images to be used, as it can resize them if needed, and most importantly the path to the images to indicate the place from where the images are to be loaded. It can be a general path to the images as the algorithm used by the method automatically attributes classes to the images according to the way the folders are set up in that path with the dataset. Other methods can for instance take a CSV (Comma-Separated Values) file with the paths to the files and use it as a guide to know where to get the files. At least 2 data generators must be created (one for train end one for test) and can be defined before or after creating the model but must be defined before the training as the training uses a special method called fit_generator. There, the settings to train the model such as epochs are entered as well as the generators and callbacks. The batch size is entered whilst calling the flow_from_directory method. Several settings were adjusted throughout the experiments with this model, such as settings in the data generators as well as epochs and batch size.

After the training was complete, the plots of accuracy and loss were obtained as well as confusion matrices to help evaluate the model performance later.

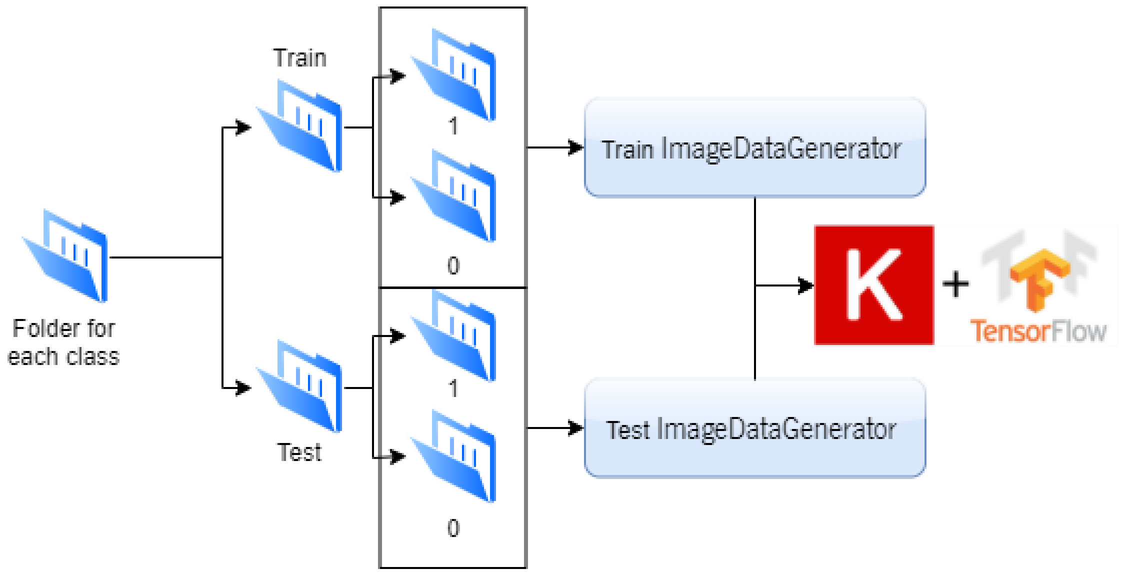


Figure 4-4. Loading using ImageDataGenerators.

4.3.3 MODEL EVALUATION

As discussed before in this dissertation, there are several ways to measure the quality of a model. Several of these measures of the quality of classification are built from a confusion matrix. This matrix records correctly and incorrectly recognized examples for each class [61]. Most importantly, it gives insight not only into the errors being made by the model but also into the types of errors that are being made by it. By having access to the type of errors being made, the model can be more easily adapted to overcome the problem. This type of matrix can be constructed in several ways, from 2x2 matrix when only two classes exist to whatever size suits the needs of the model. In this work, 5x5 confusion matrices were constructed initially as there are 5 classes of atrophy. However, it eventually came to be only 2x2 matrices as the models where simplified to classify each atrophy stage as existing or not in the study. A 2x2 confusion matrix can assign for example the columns for the predicted and the rows for the true value of the output (Table 4-3).

Table 4-3. Confusion matrix for tumor classification (binary).

		Predicted label	
		Has that atrophy stage	Other atrophy stage
True label	Has that atrophy stage	TP	FN
	Other atrophy stage	FP	TN

The combination of columns and rows of this specific type confusion matrix results in:

- True Positive – TP: values correctly predicted as True.
- True Negative – TN: values correctly predicted as False.
- False Positive – FP: values incorrectly predicted as True. FP is also called type I error.
- False Negative – FN: values incorrectly predicted as False. FN is also called type II error.

Several metrics can then be calculated from this confusion matrix. Accuracy, for instance, is defined as:

$$accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{TP + FP + FN + TN} \quad (1)$$

To evaluate the model, just like the training step, special methods were used due to the data generators; the process is illustrated in Figure 4-5. The "evaluate_generator" method was used to evaluate the model and get its accuracy and afterwards the "predict_generator" method, close to the predict method but for data generators, was used in order to get a prediction from the model.

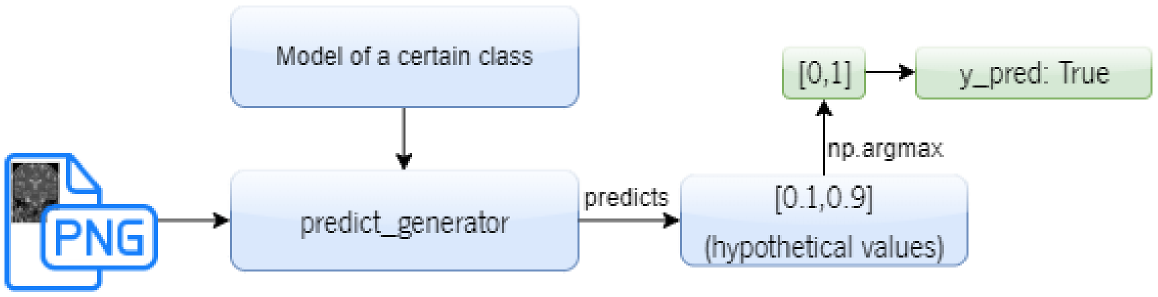


Figure 4-5. Illustration of how labels are predicted by the predict_generator.

A prediction using this model outputs the probability for each of the output classes (True or False) to be present in the image. Using argmax in that prediction can then identify which of the outputs has a higher probability and as such determine if the atrophy stage represented by that model was, or not, present in the image depending on the higher probability output. This is done for each of the atrophy classes where for every atrophy class the image is predicted to be True (has that stage of atrophy) or False (doesn't have that stage).

In the end, to understand the model's performance more clearly, a confusion matrix was constructed, for each of the models, to better understand the meaning of the predicted data. These confusion matrices compare the true labels and the predicted labels (Figure 4-6). In them, the number of correctly predicted labels can be identified.

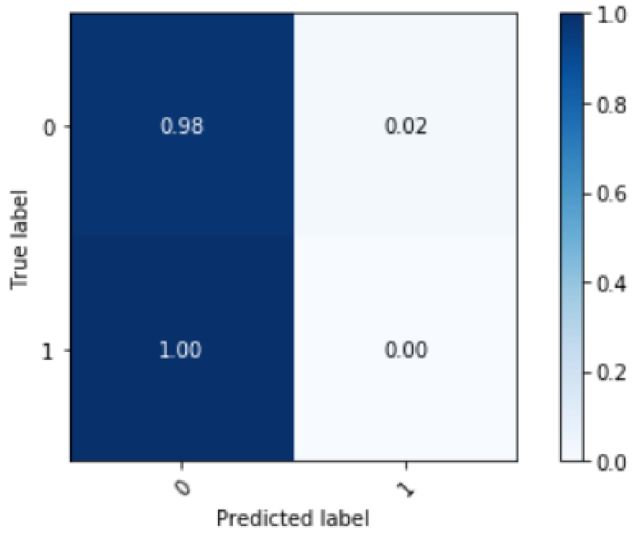
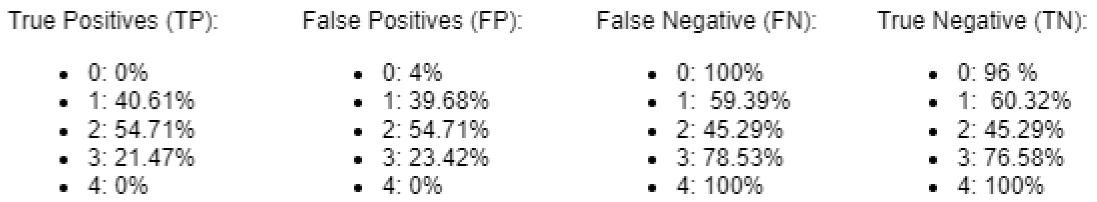


Figure 4-6. Normalized confusion matrix corresponding to class 0 model.

As can be analyzed in the figure, the results were partially good. As 98% of the Images classified as 0 for the class 0 were correctly predicted. The True label and predicted label should coincide to obtain good results and if that was the case, the matrix would show a diagonal with the value 1. This would mean 100% of the samples were predicted correctly both the ones that have that atrophy stage as well as the ones that do not.
The remaining classes had similar results were the class 2 had the most balanced results.

4.4 RESULTS AND DISCUSSION

Accuracy usually represents a good metric but as discussed before this is not always the case. Again, in the Brain Atrophy models the accuracy was not the best metrics as in, for example, the class 0 model, an accuracy of 95.94% was achieved. This value is, however, meaningless when analyzing its confusion matrix (Figure 4-6). As the dataset was very unbalanced, only a few cases represented a true class 0, which were all badly classified by the model. However, as the model correctly predicted the studies which did not have the class 0, a high accuracy was achieved even though the model didn't learn as it should.
Although the results obtained by these models weren't the expected, it got some interesting results nonetheless.
The Figure 4-7 shows the percentages of True Positive, True negative, False Positive and False Negative obtained by the models and normalized to present a percentage. The TP and FN should sum up to 100% as well as the FP with the TN. As can be seen some of the models where more successful than others but all failed at some point.



True Positives (TP):	False Positives (FP):	False Negative (FN):	True Negative (TN):
0: 0%	0: 4%	0: 100%	0: 96 %
1: 40.61%	1: 39.68%	1: 59.39%	1: 60.32%
2: 54.71%	2: 54.71%	2: 45.29%	2: 45.29%
3: 21.47%	3: 23.42%	3: 78.53%	3: 76.58%
4: 0%	4: 0%	4: 100%	4: 100%

Figure 4-7. Percentage of TP, TN, FP and FN of the Brain Atrophy models.

Even though they didn't classify the images correctly in most cases, as seen in Figure 4-8 and Figure 4-9, GradCam was applied to the models to observe, in a way, what the model found interesting in the images to classify them as it did. As seen on the referred images, the models couldn't produce every type of classification, i.e. didn't get all the TP, FP, etc. They did, however, show some type of consistency at where the model 'looked' for features. In several images the model looked at the outside of the brain area, an explanation for this can be, the model is trying to compare blank space in the brain tissue with the external space. Two sets of images where applied to observe if the model was consistent. As seen below, the results of the Brain Atrophy model were partially good, with more balanced data, better results could be achieved by the model in the future

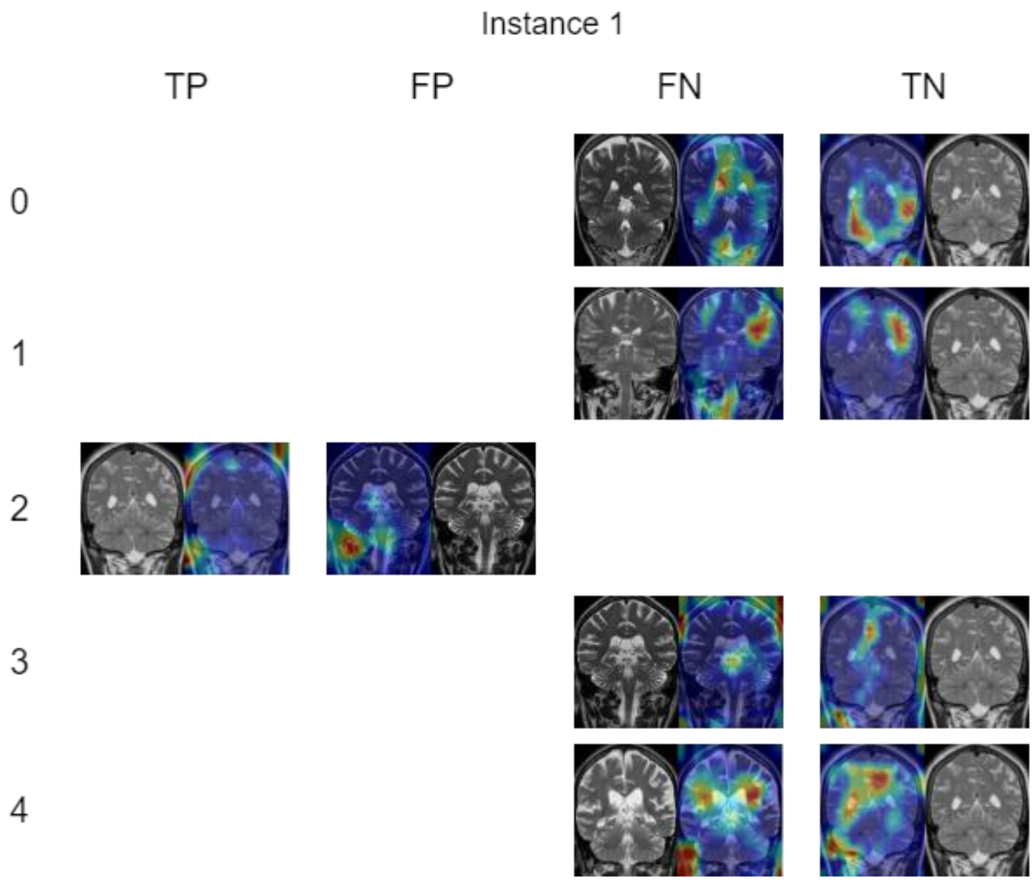


Figure 4-8. GradCam applied to the models of the Brain Atrophy - first set of images.

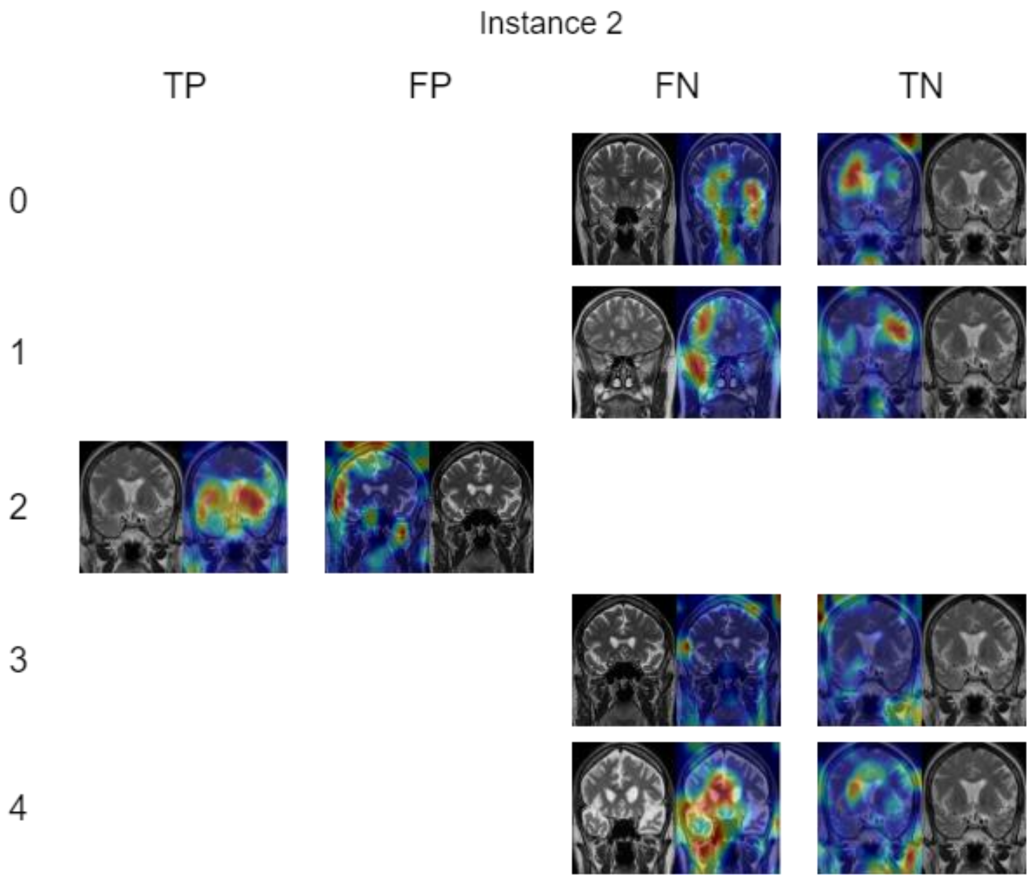


Figure 4-9. GradCam applied to the models of the Brain Atrophy - second set of images.

5 CONCLUSIONS

This final section of the dissertation concludes it by summarizing what was discussed throughout this work. It then continues by discussing the obtained results as well as possible ideas to improve this work if continued in the future.

5.1 SUMMARY

Summing up the topics depicted in this dissertation, it started with an Introduction to the main topics and described the problem with the brain spatial normalization as it is obtained nowadays and proposes a new way to perform the spatial normalization, a Deep Artificial Neural Network. A detailed description of the Technologies and Concepts involved in the creation of the network to attempt to outperform the existing normalization tools was then made. It starts by contextualizing the MRI, the main technology involved as it provides the brain studies data that need to undergo the normalization procedure. Next, the process of normalization itself was explained to better understand how beneficial a new approach would be to solve the problems of the existing one. The XNAT was then introduced to understand the benefits of having a secure archive to store the brain studies. It was also explained what Artificial Neural Networks are and the environment in which they were implemented to allow the replication of results in the future.

As the main contributions of this work, the approaches made were then better described. Spatial Normalization of the Brain explained the process of creating the Deep Artificial Neural Network and the results achieved by each architecture experimented while trying to achieve a better normalization of the data. Brain Atrophy Classification then described the second objective of the dissertation, the creation of a Deep Artificial Neural Network to predict the stage of Brain Atrophy of the presented studies. It described the approach made and results obtained with the network used. Finally, conclusions are made as well as an identification of future work to achieve usable results.

5.2 CONCLUSION

The main goal for this work was the development of Deep Learning tools to improve already existing solutions. One of these tools tried to solve the problem of spatial normalization whereas the second tool sought to assist clinical professionals when trying to evaluate their patients' brain atrophy state. Although seemingly different areas they have much in common throughout this work.

To conclude this work and develop the desired tools, knowledge of several technologies had to be acquired. Firstly, the MRI technology had to be understood in order to properly handle the MRI studies in both datasets. In the case of spatial normalization, the process itself and the tools involved in it had to be understood and in the case of the brain atrophy, the condition had to be understood too in order to correctly create DL models to perform the necessary tasks. After understanding the MRI, spatial normalization and brain atrophy, knowledge in Deep Artificial Neural Networks had to be acquired to create proper architectures which had to be tailored to solve the problems in hands. Several architectures were tried to approximate from the desired output. Throughout this work, access to a machine with a very powerful GPU was indispensable as common GPUs would not be capable of handling the quantity of information and processing involved. As such dealing with said machine and with the Deep Learning technologies was very enriching and interesting.

Furthermore, XNAT was also a key module in this work as it allowed easy access to the MRI studies both in download as well as in upload. The container used as well as Jupyter Notebooks contained in it were also indispensable where the Python modules could be developed without interferences and with an user-friendly UI.

Regarding the Spatial Normalization approach made, the results were very promising where, especially in the approach using a U-Net based architecture, the shape was accomplished correctly having a dice loss value of 0.00313 at the final stage of training, meaning 99.687% (0.99687) of the predicted output was overlapped with the expected image, i.e. an almost perfect shape was predicted. With some more modifications made to the model it could achieve even better results. For the time spent and data available to train the model the results were very good. The GAN network did not achieve results as good as the U-Net, but the shape was correctly approximated nonetheless.

Concerning the brain atrophy model, the results were also partially good, where part of the classes where correctly predicted. The results obtained are not perfect as, for instance, the class 0 was hard to classify but, as there was not that much time to develop both the spatial normalization and the brain atrophy models the results are very good. In both cases, more time invested into the development of the model and better data (less unbalanced in the brain atrophy and more data in both cases) the results achieved would outperform the current ones.

In brief, the results generated by this work, in both approaches, shows great potential and shows that these problems can indeed be solved if some more effort is put into developing a solution.

5.3 FUTURE WORK

Looking at the results obtained, although very promising, there is still a lot to be improved and optimized. For the Brain Spatial Normalization section of this work, a bigger dataset for example, would probably help achieve better results, however, the amount of memory to process that higher amount of data could be considerable. As such, methods like the Image Data Generator used in the Brain Atrophy network would be beneficial. If an even better machine was available to compute the network, a different architecture could also be tried. Changing the architecture would probably have a great impact in the results if the architecture involved 3-dimensional convolutions instead of the 2-dimensional ones used due to lack of resources. Even though the machine used was way above average, it had its limitations and couldn't compute an architecture with that kind of complexity. The 3-dimensional version of the convolutions would help evaluate the MRI studies used as a hole instead of looking at them slice by slice. This would probably be what would impact the results the most as this way the model process would be closer to the actual normalization process where the study is arranged until it matches the reference image.

If a better machine could not be used, the model settings could also be changed again to reach better results. The pre-processing step could also be improved, where the step of normalizing the data to be between 0 and 1 could be done in a different way. The approach made in this work was to divide each study by its maximum value. Although this works, it doesn't guarantee the data is normalized properly as the studies had very different maximum values and a wide range of intensities throughout the studies themselves. A possible better way to normalize the values would be to divide all the studies by the same value, higher than the maximum value of all the studies. This would probably make the max normalized values would not reach the value 1 but this way the data is all normalized equally. Other approach would be to also include an intensity normalization of the studies.

As seen on the results of the U-Net based architecture, this architecture could be revisited in the future using the changes discussed before as it presented the best results of the architectures tried.

As seen on Figure 5-1, which is part of Figure 3-5, we can see the U-Net based architecture got the main shapes right as they can be recognized in the right most image. This shows that, with some of the modifications mentioned, the architecture could improve its results.

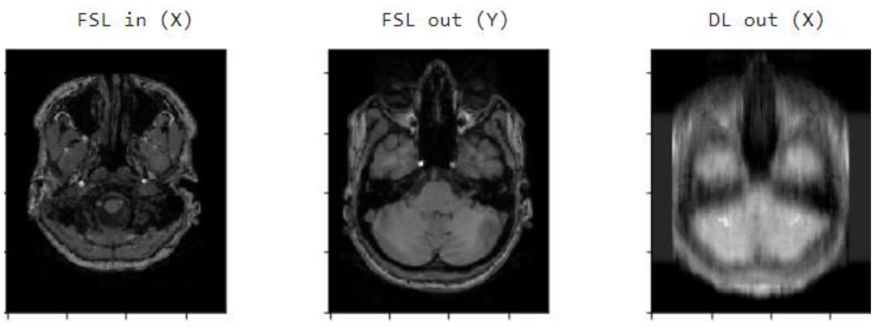


Figure 5-1. Results obtained by the U-Net based architecture.

One last possible approach to make would be to use the model not to predict the normalized image but to predict a warp matrix as used by the FSL. This way, the information to predict would be less complex and the warp matrix could then be used in FSL to perform the normalization still in a much faster way than the existing solution.

Regarding the brain atrophy section, several improvements can also be made. Firstly, the dataset provided, although containing a high number of images, could not be used in its entireness. Several planes were provided for each patient in the dataset but only one type of plane could be used. The first thing to try in a future work would be to use a better dataset containing a higher number of samples and all in the same plane. This would insure a good percentage of the studies in the dataset was used and help improve results. The other disadvantage of the dataset used was the unbalance present. As the classes were fairly unbalanced a model was instead trained for each class. Doing this improved the results but was not enough to produce the desired outcome, as the classes were still very unbalanced, even when classifying the studies regarding the presence or not of that specific class. Therefore, a more balanced dataset could improve the results. Finally, in the topic of the dataset used, one more thing that had a great impact in the results was the labeling method. The dataset was labeled with two labels for each patient, corresponding to the presence of atrophy in each of the patients' half of the brain. This becomes challenging as it would not be beneficial to divide the studies in half and label each half. On the other hand, by having two labels, only one can be used, and the label used can be wrongly chosen. To overcome this the data was duplicated in the cases where the two labels were different and one label was attributed to each of the copies, and the cases where the labels where the same, it was simply attributed to the corresponding image. In a future work, if the same dataset had to be used, it could hopefully be labeled in a better way, using only one label per study and not per patient. By using one label per patient like the dataset used, doesn't guarantee the atrophy is present in every slice of the patient's brain. This means, a better evaluation of the dataset by a professional would help to maintain the integrity and validity of the information.

After solving the dataset problems, several settings can be changed in the model to hopefully achieve a better result. One of the changes that could be mad would be to, like in the spatial normalization change the convolutions to 3-dimensional convolutions which would make sure the model considers the image in its entirety.

In the end, when both the Spatial Normalization and the Brain atrophy models work as intended, a possible future work would be to implement an user-friendly UI (User Interface) to easily use them. This UI could, for instance, be integrated in the XNAT server or created in an independent way.

REFERENCES

[1] R. Haux, “Medical informatics: Past, present, future,” *Int. J. Med. Inform.*, vol. 79, no. 9, pp. 599–610, Sep. 2010.

[2] A. P. James and B. V. Dasarathy, “Medical image fusion: A survey of the state of the art,” *Inf. Fusion*, vol. 19, no. 1, pp. 4–19, 2014.

[3] W. M. Wells, “Medical Image Analysis - past, present, and future.,” *Med. Image Anal.*, vol. 33, pp. 4–6, Oct. 2016.

[4] C. J. Price and K. J. Friston, “Scanning patients with tasks they can perform,” *Hum. Brain Mapp.*, vol. 8, no. 2–3, pp. 102–108, Jan. 1999.

[5] J. S. George *et al.*, “Mapping function in the human brain with magnetoencephalography, anatomical magnetic resonance imaging, and functional magnetic resonance imaging.,” *J. Clin. Neurophysiol.*, vol. 12, no. 5, pp. 406–31, Sep. 1995.

[6] F. Chollet, *Deep Learning with Python*. Manning Publications Co., 2018.

[7] N. Buduma, *Fundamentals of Deep Learning : Designing Next-Generation Machine Intelligence Algorithms*, vol. 44, no. 5. 2017.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016.

[9] Nvidia, “The Difference Between AI, Machine Learning, and Deep Learning?,” 2016. [Online]. Available: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/. [Accessed: 15-Oct-2018].

[10] F. Altaf, S. M. S. Islam, N. Akhtar, and N. K. Janjua, “Going Deep in Medical Image Analysis: Concepts, Methods, Challenges and Future Directions,” 2019.

[11] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A Design Science Research Methodology for Information Systems Research,” 2007.

[12] S. T. March and V. C. Storey, “DESIGN SCIENCE IN THE INFORMATION SYSTEMS DISCIPLINE: AN INTRODUCTION TO THE SPECIAL ISSUE ON DESIGN SCIENCE RESEARCH,” 2008.

[13] N. Ibtehaz and M. S. Rahman, “MultiResUNet : Rethinking the U-Net Architecture for Multimodal Biomedical Image Segmentation,” 2019.

[14] J. Q. Azasoo and K. O. Boateng, “A Retrofit Design Science Methodology for Smart Metering Design in Developing Countries,” in *Proceedings - 15th International Conference on Computational Science and Its Applications, ICCSA 2015*, 2015, pp. 1–7.

[15] D. W. McRobbie, E. A. Moore, M. J. Graves, and M. R. Prince, *MRI From Picture to Proton*, 2nd ed. Cambridge: CAMBRIDGE UNIVERSITY PRESS, 2006.

[16] R. W. Brown, Y.-C. N. Cheng, E. M. Haacke, M. R. Thompson, and R. Venkatesan, *Magnetic resonance imaging : physical principles and sequence design*, 2nd ed. 2014.

[17] A. Berger, "Magnetic resonance imaging.," *BMJ*, vol. 324, no. 7328, p. 35, Jan. 2002.

[18] C. A. Roobottom, G. Mitchell, and G. Morgan-Hughes, "Radiation-reduction strategies in cardiac computed tomographic angiography," 2010.

[19] J. E. Chen and G. H. Glover, "Functional Magnetic Resonance Imaging Methods."

[20] A. D. Elster, "How does fMRI work?" [Online]. Available: https://www.mriquestions.com/how-does-fmri-work.html. [Accessed: 28-Aug-2019].

[21] A. D. Elster, "What is Diffusion Tensor Imaging (DTI), and how does it differ from 'regular' diffusion-weighed imaging?" [Online]. Available: https://www.mriquestions.com/dti-tensor-imaging.html. [Accessed: 28-Aug-2019].

[22] D. K. Jones and P. J. Basser, "Diffusion-tensor MRI: theory, experimental design and data analysis - a technical review," *NMR Biomed.*, 2002.

[23] L. MINATI and W. Weglarz, "Physical Foundations, Models, and Methods of Diffusion Magnetic Resonance Imaging of the Brain: A Review," in *Concepts in Magnetic Resonance Part A, Vol. 30A(5)* , Wiley InterScience, 2007, pp. 278–307.

[24] R. A. Poldrack, J. A. Mumford, and T. E. Nichols, "Spatial normalization," in *Handbook of functional MRI data analysis*, Cambridge University Press, 2011, pp. 53–69.

[25] R. S. Snell, *CLINICAL ANATOMY BY REGIONS*, 9th ed. Lippincott Williams & Wilkins, 2012.

[26] A. D. Elster, "What is the difference between co-registration and normalization? How are these performed?" [Online]. Available: http://mriquestions.com/registrationnormalization.html. [Accessed: 09-Oct-2018].

[27] M. Brett, K. Christoff, R. Cusack, and J. L. Lancaster, "Using the Talairach Atlas with the MNI template The Talairach atlas."

[28] M. W. Woolrich *et al.*, "Bayesian analysis of neuroimaging data in FSL.," *Neuroimage*, vol. 45, no. 1 Suppl, 2009.

[29] "FSL - FslWiki." [Online]. Available: https://fsl.fmrib.ox.ac.uk/fsl/fslwiki. [Accessed: 18-Nov-2018].

[30] M. Jenkinson, C. F. Beckmann, T. E. J. Behrens, M. W. Woolrich, and S. M. Smith, "FSL," *Neuroimage*, vol. 62, no. 2, pp. 782–790, Aug. 2012.

[31] "BET - FslWiki." [Online]. Available: https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/BET. [Accessed: 18-Nov-2018].

[32] S. M. Smith, “Fast robust automated brain extraction,” *Hum. Brain Mapp.*, vol. 17, no. 3, pp. 143–155, Nov. 2002.

[33] “FLIRT - FslWiki.” [Online]. Available: https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FLIRT. [Accessed: 18-Nov-2018].

[34] M. Jenkinson and S. Smith, “A global optimisation method for robust affine registration of brain images,” *Med. Image Anal.*, vol. 5, no. 2, pp. 143–156, Jun. 2001.

[35] “FNIRT - FslWiki.” [Online]. Available: https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FNIRT. [Accessed: 18-Nov-2018].

[36] K. A. Archie and D. Marcus, “Head in the cloud: accessing distributed data and services through XNAT,” in *Frontiers in Neuroinformatics 7*, 2013.

[37] D. S. Marcus, T. R. Olsen, M. Ramaratnam, and R. L. Buckner, “The extensible neuroimaging archive toolkit,” *Neuroinformatics*, vol. 5, no. 1, pp. 11–33, Mar. 2007.

[38] Y. Schwartz *et al.*, “PyXNAT: XNAT in Python,” *Front. Neuroinform.*, vol. 6, no. 12, 2012.

[39] R. Moreira, “Github - rgllm/xnatum.” [Online]. Available: https://github.com/rgllm/xnatum. [Accessed: 06-Sep-2019].

[40] D. B. Keator *et al.*, “A National Human Neuroimaging Collaboratory Enabled by the Biomedical Informatics Research Network (BIRN),” *IEEE Trans Inf Technol Biomed*, vol. 12, no. 2, pp. 162–172, 2008.

[41] H. H. M. I. Washington University School of Medicine, Harvard University, “XNAT Documentation.” [Online]. Available: http://xnat.org/about/. [Accessed: 06-Sep-2019].

[42] A. Maier, C. Syben, T. Lasser, and C. Riess, “A gentle introduction to deep learning in medical image processing,” *Z. Med. Phys.*, Jan. 2019.

[43] M. A. Nielsen, “Neural Networks and Deep Learning.” Determination Press, 2015.

[44] Q. Liu and M. Fu, “Dice Loss Function for image Segmentation Using Dense Dilation Spatial Pooling Network,” 2018.

[45] L. Fidon *et al.*, “Generalised Wasserstein Dice Score for Imbalanced Multi-class Segmentation using Holistic Convolutional Networks.”

[46] J. Patterson and A. Gibson, *Deep learning : a practitioner’s approach.* .

[47] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, “3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation,” Jun. 2016.

[48] R. Khandelwal, “Deep Learning — Generative Adversarial Network(GAN).” [Online]. Available: https://medium.com/datadriveninvestor/deep-learning-generative-adversarial-network-gan-

34abb43c0644. [Accessed: 25-Sep-2019].

[49] I. J. Goodfellow *et al.*, “Generative Adversarial Nets.”

[50] S. M. Smith *et al.*, “Advances in functional and structural MR image analysis and implementation as FSL,” in *NeuroImage*, 2004, vol. 23, no. SUPPL. 1.

[51] M. Osorio, C. Buil-Aranda, and H. Vargas, “DockerPedia: a Knowledge Graph of Docker Images.”

[52] NVIDIA, “NVIDIA Docker: GPU Server Application Deployment Made Easy.” [Online]. Available: https://devblogs.nvidia.com/nvidia-docker-gpu-server-application-deployment-made-easy/. [Accessed: 20-Jun-2019].

[53] J. Weber, “Idiomatic and Reproducible Software Builds using Containers for Reliable Computing,” 2016.

[54] B. M. Randles, I. V. Pasquetto, M. S. Golshan, and C. L. Borgman, “Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study,” in *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, 2017, pp. 1–2.

[55] P. Pandey, “Jupyter Lab: Evolution of the Jupyter Notebook,” 2019. [Online]. Available: https://towardsdatascience.com/jupyter-lab-evolution-of-the-jupyter-notebook-5297cacde6b. [Accessed: 22-Jul-2019].

[56] J. E. Gaugler, H. Ascher-Svanum, D. L. Roth, T. Fafowora, A. Siderowf, and T. G. Beach, “Characteristics of patients misdiagnosed with Alzheimer’s disease and their medication use: An analysis of the NACC-UDS database,” *BMC Geriatr.*, vol. 13, no. 1, Dec. 2013.

[57] L. Harper *et al.*, “MRI visual rating scales in the diagnosis of dementia: Evaluation in 184 post-mortem confirmed cases,” *Brain*, vol. 139, no. 4, pp. 1211–1225, Apr. 2016.

[58] M. Jorge Cardoso *et al.*, “STEPS: Similarity and Truth Estimation for Propagated Segmentations and its application to hippocampal segmentation and brain parcelation,” *Med. Image Anal.*, vol. 17, no. 6, pp. 671–684, Aug. 2013.

[59] J. Schrouff *et al.*, “PRoNTo: Pattern recognition for neuroimaging toolbox,” *Neuroinformatics*, vol. 11, no. 3, pp. 319–337, Jul. 2013.

[60] D. M. St-Amant, A. P. F. Gaillard, and et al., “Medial temporal lobe atrophy score | Radiology Reference Article | Radiopaedia.org.” [Online]. Available: https://radiopaedia.org/articles/medial-temporal-lobe-atrophy-score?lang=us. [Accessed: 13-Nov-2019].

[61] M. Sokolova, N. Japkowicz, and S. Szpakowicz, “Beyond Accuracy, F-Score and ROC: A

Family of Discriminant Measures for Performance Evaluation," pp. 1015–1021, 2006.

[62] M. Brett *et al.*, "nibabel: 2.4.1," May 2019.

APPENDICES

A – FULL LIST OF USED LIBRARIES

To allow replication of the results obtained in this work, the full list of modules used, and respective versions are discriminated below in the Table A-1. The settings of the machine used are also described in sections 3.2 and 4.2 to fully replicate the results.

Table A-1. Table of conda-list and pip list full information.

Package name	Package version	Package name	Package version
absl-py	0.7.1	nb_conda	2.2.1
alabaster	0.7.12	nb_conda_kernels	2.2.0
ann-visualizer	2.5	nbconvert	5.3.1
asn1crypto	0.24.0	nbformat	4.4.0
astor	0.7.1	ncurses	6.1
attrs	18.2.0	networkx	2.3
babel	2.6.0	nibabel	2.4.1
backcall	0.1.0	ninja	1.8.2
backports	1.0	notebook	5.7.4
backports.os	0.1.1	numexpr	2.6.9
bcolz	1.2.1	numpy	1.15.4
beautifulsoup4	4.6.3	numpy-base	1.15.4
blas	1.0	nvidia-ml-py3	7.352.0
bleach	3.1.0	olefile	0.46
blosc	1.15.0	opencv-python	4.0.0.21
bokeh	1.0.4	openssl	1.1.1b
bottleneck	1.2.1	packaging	18.0
bzip2	1.0.6	pandas	0.24.0
ca-certificates	2019.1.23	pandas-summary	0.0.5
certifi	2019.3.9	pandoc	2.2.3.2
cffi	1.11.5	pandocfilters	1.4.2
chardet	3.0.4	parso	0.3.1
click	7.0	partd	0.3.9
cliff	2.8.2	path.py	11.5.0
cloudpickle	0.6.1	patsy	0.5.1
cmd2	0.9.7	pbr	5.1.1
colorama	0.4.1	pcre	8.42
configparser	3.7.1	pexpect	4.6.0
cryptography	2.6.1	pickleshare	0.7.5

cssselect	1.0.3	pillow	5.4.1
cuda90	1.0	pip	19.1
cuda92	1.0	plac	0.9.6
cudatoolkit	10.0.130	preshed	2.0.1
cycler	0.10.0	prettytable	0.7.2
cymem	2.0.2	progressbar2	3.34.3
cython	0.29.2	prometheus_client	0.5.0
cytoolz	0.9.0.1	prompt_toolkit	2.0.7
dask	1.0.0	protobuf	3.7.1
dask-core	1.0.0	psutil	5.4.8
dbus	1.13.6	ptyprocess	0.6.0
decorator	4.3.0	pyarrow	0.12.0
dicom2nifti	2.1.5	pycparser	2.19
dill	0.2.8.2	pydicom	1.2.2
distributed	1.25.2	pydot	1.4.1
docutils	0.14	pygments	2.3.1
entrypoints	0.3	pyhamcrest	1.9.0
expat	2.2.6	pyopenssl	18.0.0
fastai	1.0.42	pyparsing	2.3.1
fastprogress	0.1.18	pyperclip	1.7.0
feather-format	0.4.0	pyqt	5.9.2
fontconfig	2.13.0	pysocks	1.6.8
freetype	2.9.1	pytables	3.4.4
future	0.17.1	python	3.6.8
gast	0.2.2	python-dateutil	2.7.5
glib	2.56.2	python-graphviz	0.10.1
glob2	0.7	python-utils	2.3.0
gmp	6.1.2	pytorch	1.0.0
grpcio	1.20.1	pytz	2018.9
gst-plugins-base	1.14.0	pywavelets	1.0.3
gstreamer	1.14.0	pyyaml	3.13
h5py	2.9.0	pyzmq	17.1.2
hdf5	1.10.4	qt	5.9.7
heapdict	1.0.0	qtconsole	4.4.3
html5lib	1.0.1	readline	7.0
icu	58.2	regex	2018.01.10
idna	2.8	requests	2.21.0
imageio	2.5.0	scikit-image	0.15.0

imagesize	1.1.0
imgaug	0.2.9
importlib_metadata	0.7
intel-openmp	2019.1
ipykernel	5.1.0
ipython	7.2.0
ipython_genutils	0.2.0
ipywidgets	7.4.2
isodate	0.6.0
isoweek	1.3.3
jedi	0.13.2
jinja2	2.10
jpeg	9b
jsonschema	2.6.0
jupyter	1.0.0
jupyter_client	5.2.4
jupyter_console	6.0.0
jupyter_contrib_core	0.3.3
jupyter_contrib_nbextensions	0.5.1
jupyter_core	4.4.0
jupyter_highlight_selected_word	0.2.0
jupyter_latex_envs	1.4.4
jupyter_nbextensions_configurator	0.4.1
jupyterlab	0.35.4
jupyterlab_server	0.2.0
kaggle-cli	0.12.13
keras	2.2.4
keras-applications	1.0.7
keras-preprocessing	1.0.9
keras-tqdm	2.0.1
kiwisolver	1.0.1
libedit	3.1.20181209
libffi	3.2.1
libgcc-ng	8.2.0
libgfortran-ng	7.3.0
libiconv	1.15
scikit-learn	0.20.2
scipy	1.2.0
seaborn	0.9.0
send2trash	1.5.0
setuptools	39.1.0
shapely	1.6.4.post2
simplegeneric	0.8.1
sip	4.19.8
six	1.12.0
sklearn-pandas	1.8.0
snappy	1.1.7
snowballstemmer	1.2.1
sortedcontainers	2.1.0
soupsieve	1.7.1
spacy	2.0.18
sphinx	2.0.1
sphinx-rtd-theme	0.4.3
sphinxcontrib-applehelp	1.0.1
sphinxcontrib-devhelp	1.0.1
sphinxcontrib-htmlhelp	1.0.2
sphinxcontrib-jsmath	1.0.1
sphinxcontrib-qthelp	1.0.2
sphinxcontrib-serializinghtml	1.1.3
sqlite	3.26.0
statsmodels	0.9.0
stevedore	1.30.0
tblib	1.3.2
tensorboard	1.11.0
tensorflow-estimator	1.13.0
tensorflow-gpu	1.11.0
termcolor	1.1.0
terminado	0.8.1
testfixtures	6.3.0
testpath	0.4.2
thinc	6.12.1
tk	8.6.8

libpng	1.6.36	toolz	0.9.0
libsodium	1.0.16	torch	1.0.0
libstdcxx-ng	8.2.0	torchsummary	1.5.1
libtiff	4.0.10	torchtext	0.2.3
libuuid	1.0.3	torchvision	0.1.9
libxcb	1.13	tornado	4.5.3
libxml2	2.9.9	tqdm	4.29.1
libxslt	1.1.32	traitlets	4.3.2
livelossplot	0.4.1	typing	3.6.4
locket	0.2.0	ujson	1.35
lxml	4.0.0	urllib3	1.24.1
lzo	2.10	wcwidth	0.1.7
markdown	3.1	webencodings	0.5.1
markupsafe	1.1.0	werkzeug	0.15.2
matplotlib	3.0.2	wheel	0.32.3
mechanicalsoup	0.8.0	widgetsnbextension	3.4.2
mistune	0.8.4	wrapt	1.10.11
mkl	2019.1	xnat	0.3.17
mkl_fft	1.0.10	xnatum	1.0.6
mkl_random	1.0.2	xz	5.2.4
mock	2.0.0	yaml	0.1.7
msgpack-numpy	0.4.3.2	zeromq	4.2.5
msgpack-python	0.5.6	zict	0.1.3
murmurhash	1.0.1	zlib	1.2.11

B – U-NET BASED ARCHITECTURE USED

To better understand the Artificial Neural Network architecture used in this work, the Figure A-1, describes in more detail the specific U-Net based architecture used.

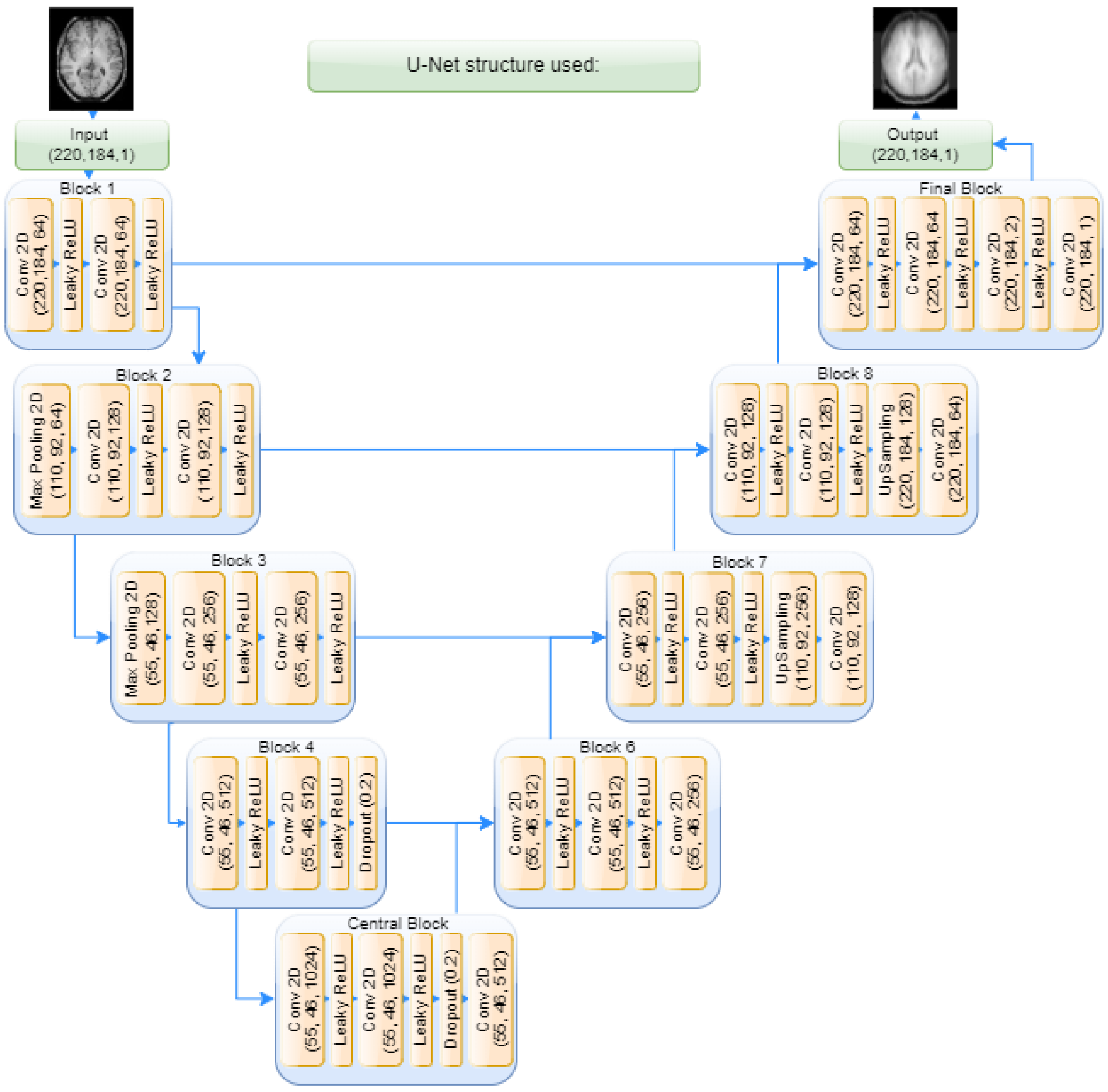


Figure A-1. U-Net based architecture used.

C – GAN ARCHITECTURE USED

To allow replication of the results obtained with the GAN network, the specific architecture used is shown below in Figure A-2.

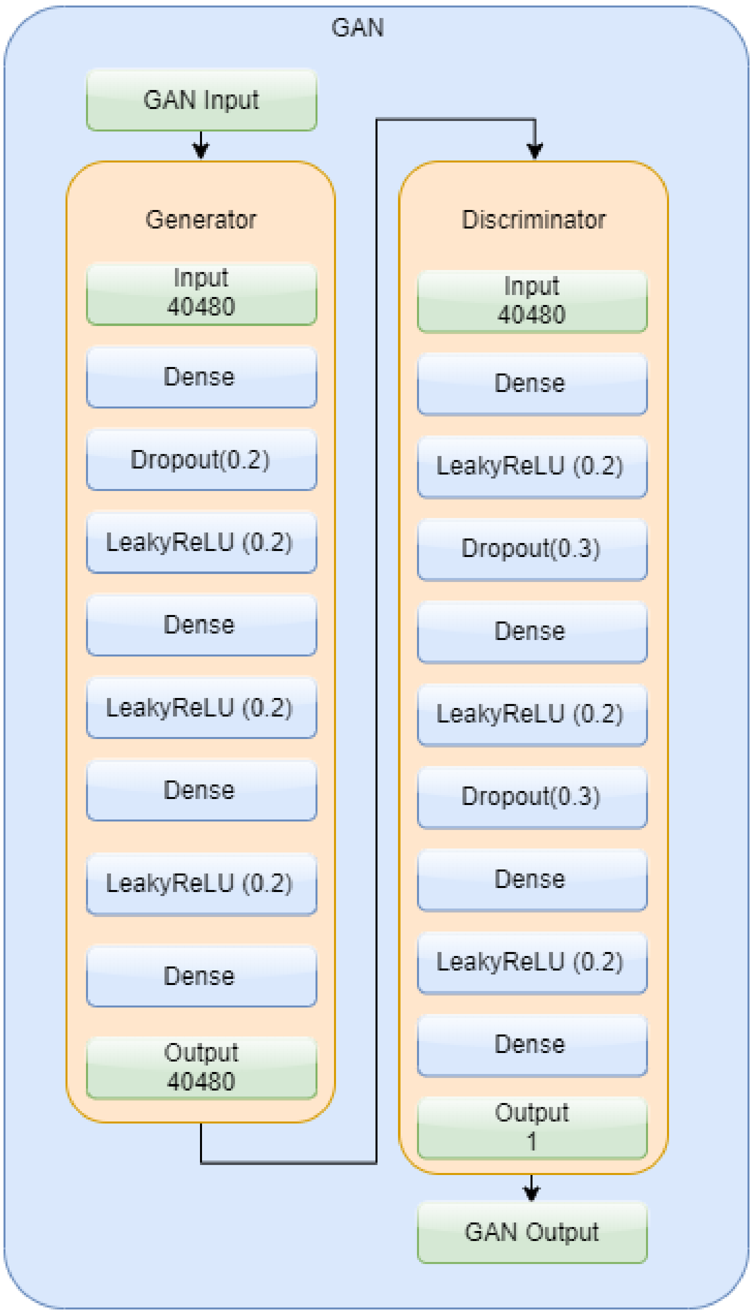


Figure A-2. General structure of the GAN used.

D – SPECIFIC CLASS BLOCKS USED IN THE BRAIN ATROPHY MODEL

The figures below, Figure A-3 - Figure A-7, represent the block used specifically for each class in the architecture described in section 4.3.2. Each figure represents the block used in the corresponding class.

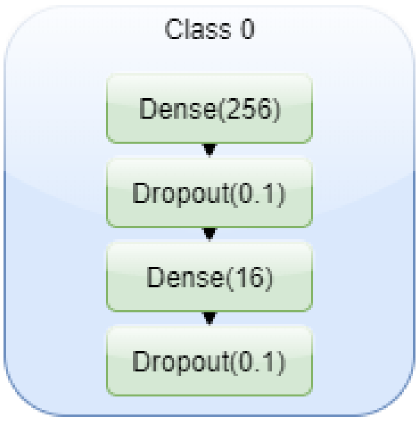


Figure A-3. Block for class 0.

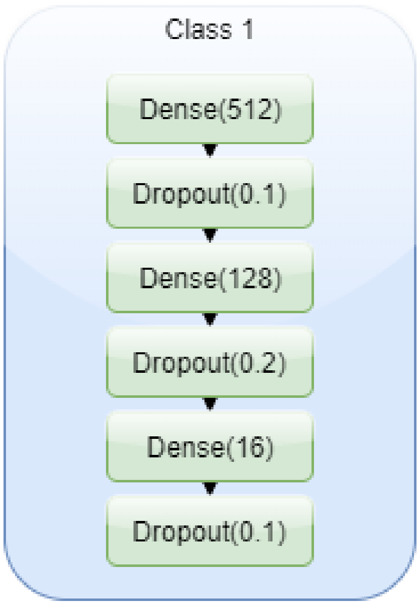


Figure A-4. Block for class 1.

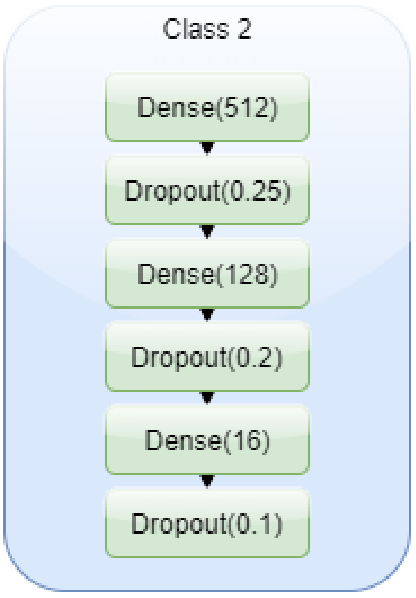


Figure A-5. Block for class 2.

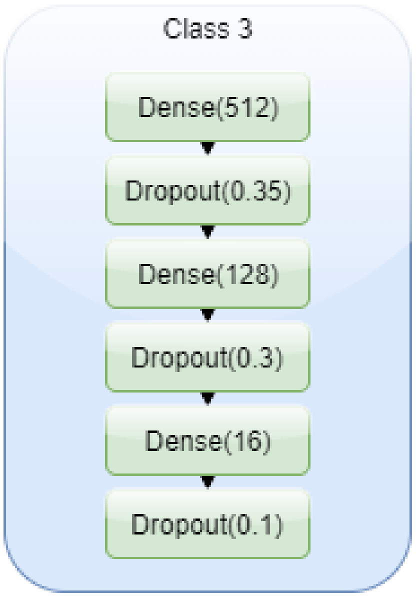


Figure A-6. Block for class 3.

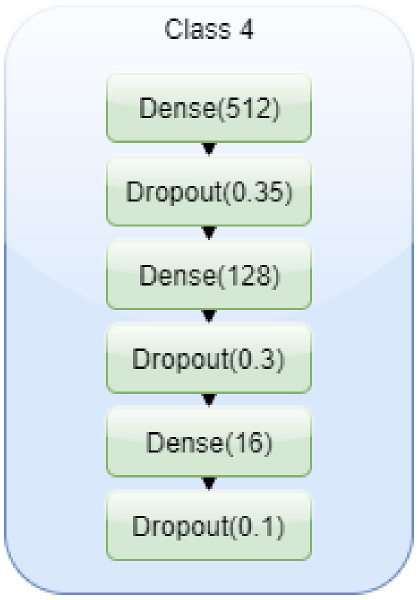


Figure A-7. Block for class 4.